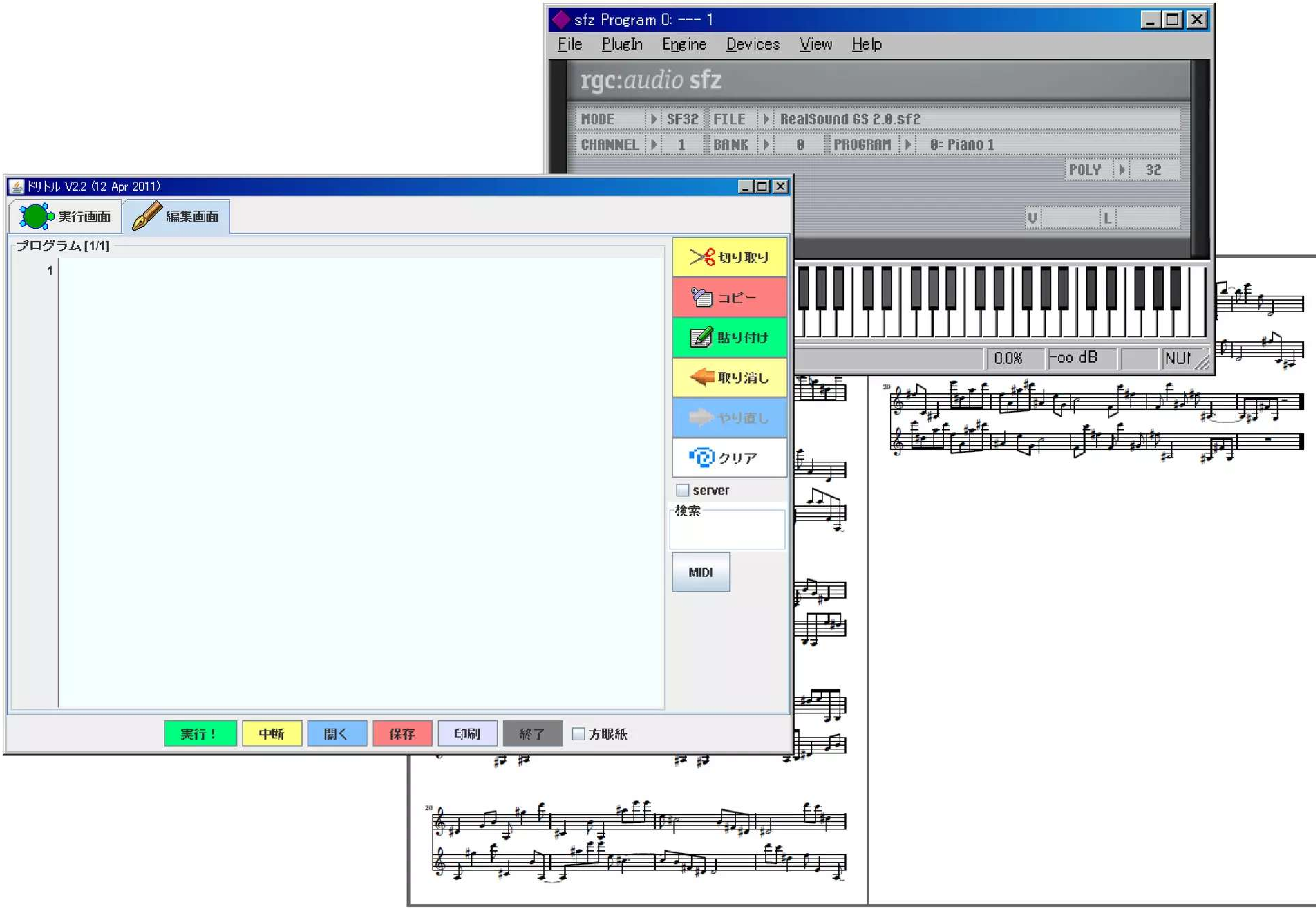
音楽プログラミング講座

- ドリトルによる実験音楽入門 -

イーテキスト研究所：山澤 昭彦



sfz Program 0: --- 1
File PlugIn Engine Devices View Help
rgc:audio sfz
MODE SF32 FILE RealSound GS 2.0.sf2
CHANNEL 1 BANK 0 PROGRAM 0: Piano 1
POLY 32
0.0% -oo dB NUM
ドリトル V2.2 (12 Apr 2011)
実行画面
編集画面
プログラム [1/1]
切り取り
コピー
貼り付け
取り消し
やり直し
クリア
server
検索
MIDI
実行！
中断
開く
保存
印刷
終了
方眼紙

I MIDI音源で音楽を演奏する

ドリトル (Dolittle) とは

- 教育用に設計されたプログラミング言語
  - 処理系は兼宗進氏（現大阪電気通信大学教授）が開発
- オブジェクト指向（プロトタイプベース）
  - あらかじめ用意されている部品を利用することで簡単にプログラミングが可能
- 名前の由来
  - Do-Little＝やることが少ない（簡単にプログラムを書ける）
- ＷＥＢサイト
  - http://dolittle.eplang.jp/
- 参考図書（ドリトルで学ぶプログラミング[第2版] ）
  - http://www.amazon.co.jp/exec/obidos/ASIN/4904013085/logob-22

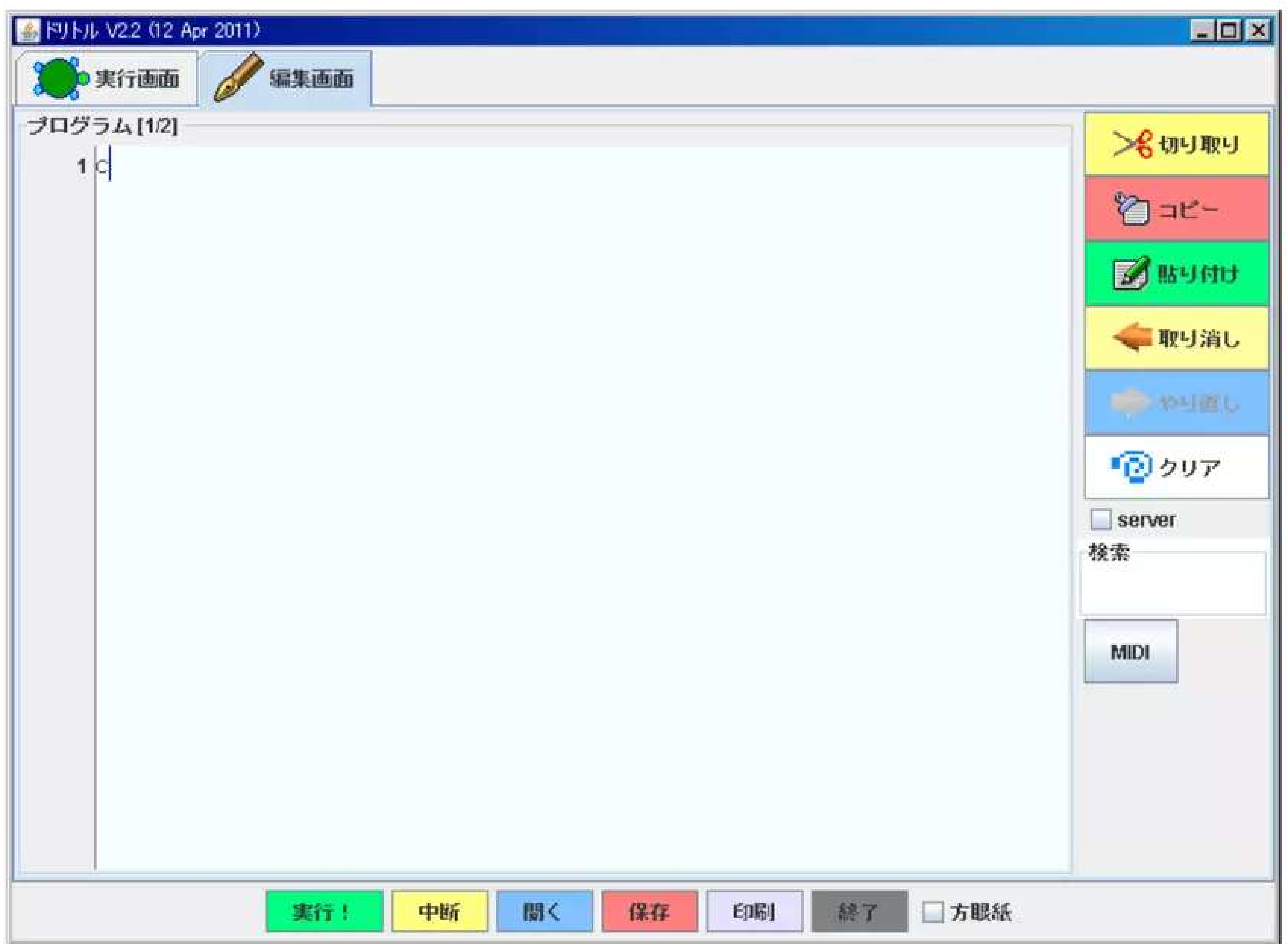


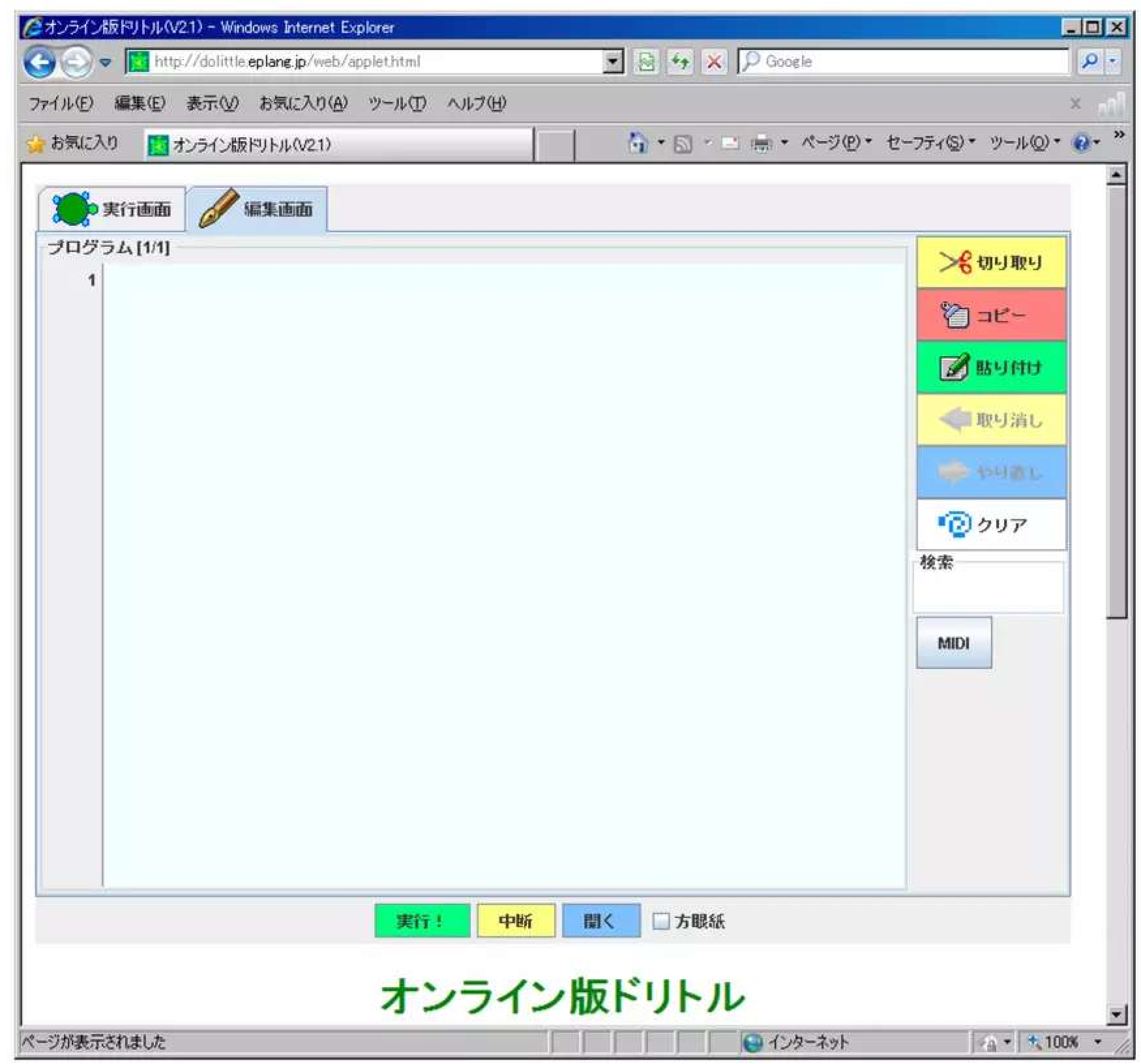
作曲にドリトルを使う理由

- フリーウェアであり誰でも無料で自由に使うことができる
- ソースプログラムの編集と実行画面が統合された開発環境
- Javaで開発されているためプラットフォームを選ばない
- Javaアプレットとしてウェブブラウザ上で動作
- オブジェクト指向言語のため音楽の要素を簡潔に表現できる
- プログラミングに日本語が使える
- 「ドレミ」のような日本語での音符の読みによる音の指定ができる
- 初心者でも簡単に音楽プログラミングができる
- MIDI音源を使ってプログラミングした内容をすぐに演奏できる
- Javaに付属している音源の他、PC内蔵音源やシンセなど外部音源も利用可能

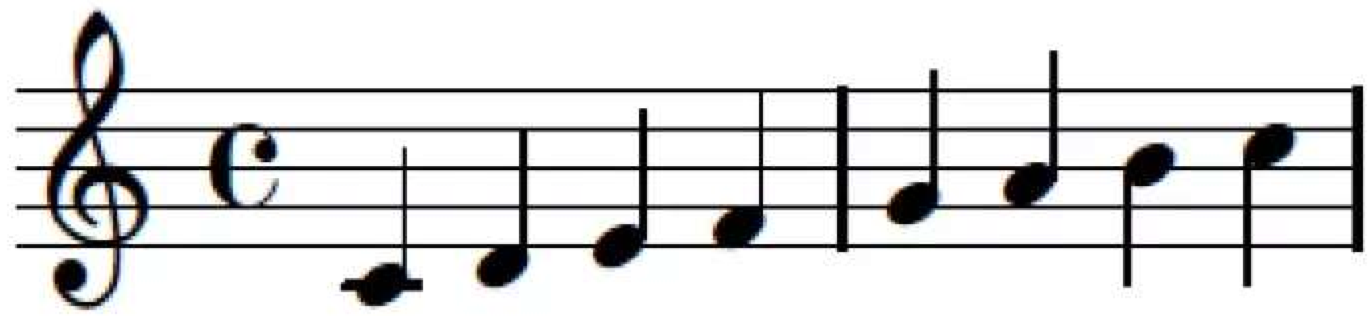
ドリトルの開発実行環境

- ローカルPCにセットアップ
- WEBサービスで簡単実行

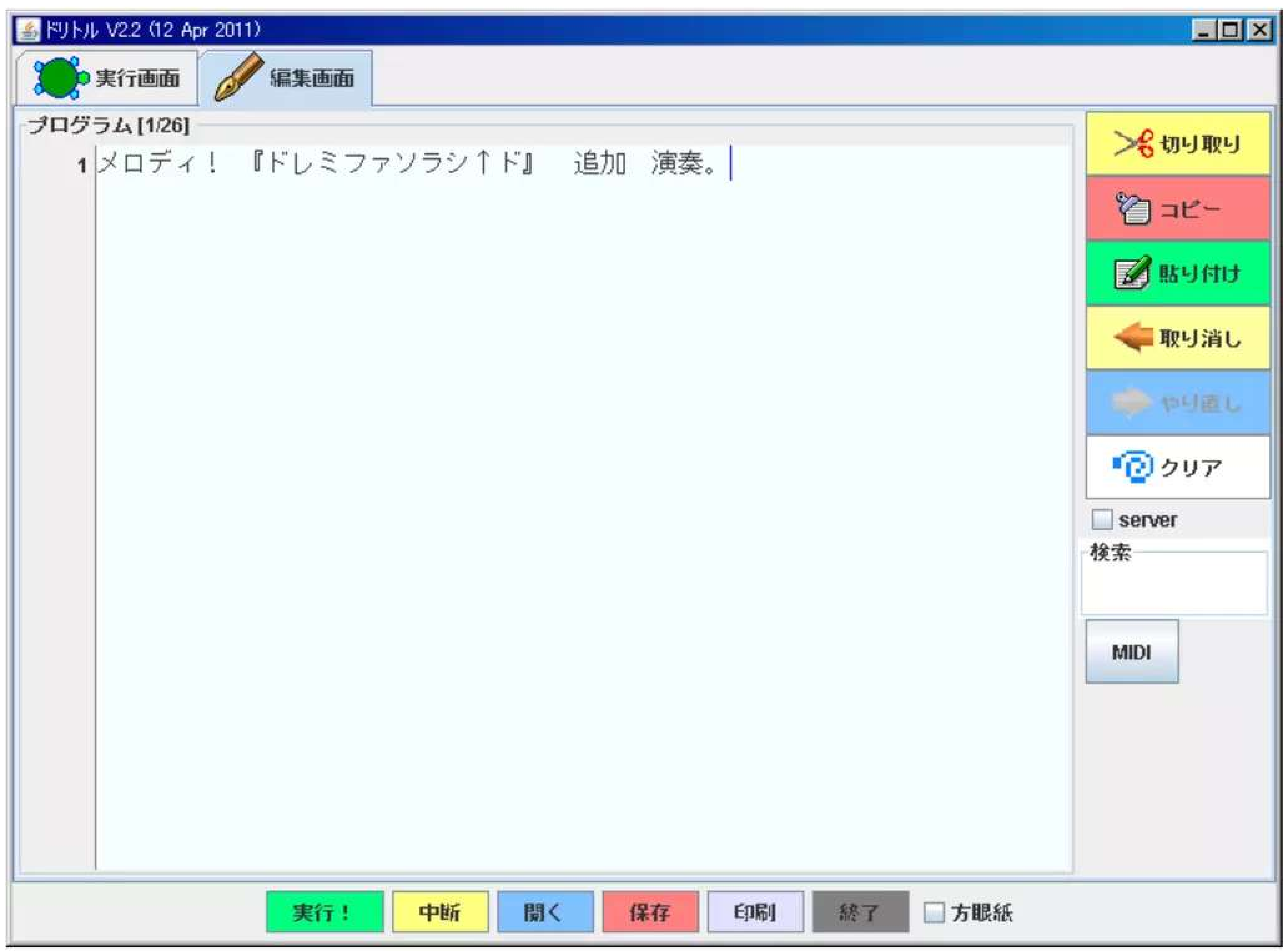




ドリトルで一行プログラム



メロディ！『ドレミファソラシ↑ド』 追加　演奏。



オブジェクトとメソッド

- リトルではメロディーオブジェクトが持っているメソッドを呼び出すことで音楽を演奏
- メソッドを呼び出すには
  - メロディ！『ドレミファソラシ↑ド』 追加　演奏。
- このような記述をメッセージ送信と呼ぶ
- ！の左側がオブジェクト、右側がメソッド
- 「メロディー」は音楽機能を実現するドリトルのオブジェクト
- 「追加」と「演奏」は「メロディー」オブジェクトのメソッド
- 『ドレミファソラシ↑ド』は「追加」メソッドのパラメータ
- 「追加」メソッドのパラメータはドリトルのＭＭＬで記述
- ＭＭＬは文字列なので『』で囲む

メッセージのカスケード

- メソッドは実行後に何らかの値（オブジェクト）を返す
- メソッドは実行後に別のメソッド名を書くことができる
  - 前のメソッドが返したオブジェクトに対するメソッド呼び出し
- これをメッセージのカスケード（直列接続）と呼ぶ
  - オブジェクト！　メソッド　メソッド　・・・・　メソッド。
- 次の2つは同じ結果
  - メロディ！『ドレミファソラシ↑ド』 追加。 メロディ！ 演奏。
  - メロディ！『ドレミファソラシ↑ド』 追加　演奏。
- カスケードすることによりプログラムを簡潔に記述

MIDI音源の切り替え

- ドリトルの音楽機能はMIDI音源を使って音を出す
- 標準ではJavaが持っているMIDI音源の音が出る
- 編集画面にある[MIDI]ボタンを押すとMIIDI音源の切り替えができる
- MIDI音源を切り替えることにより演奏にソフトシンセや外部音源を使える



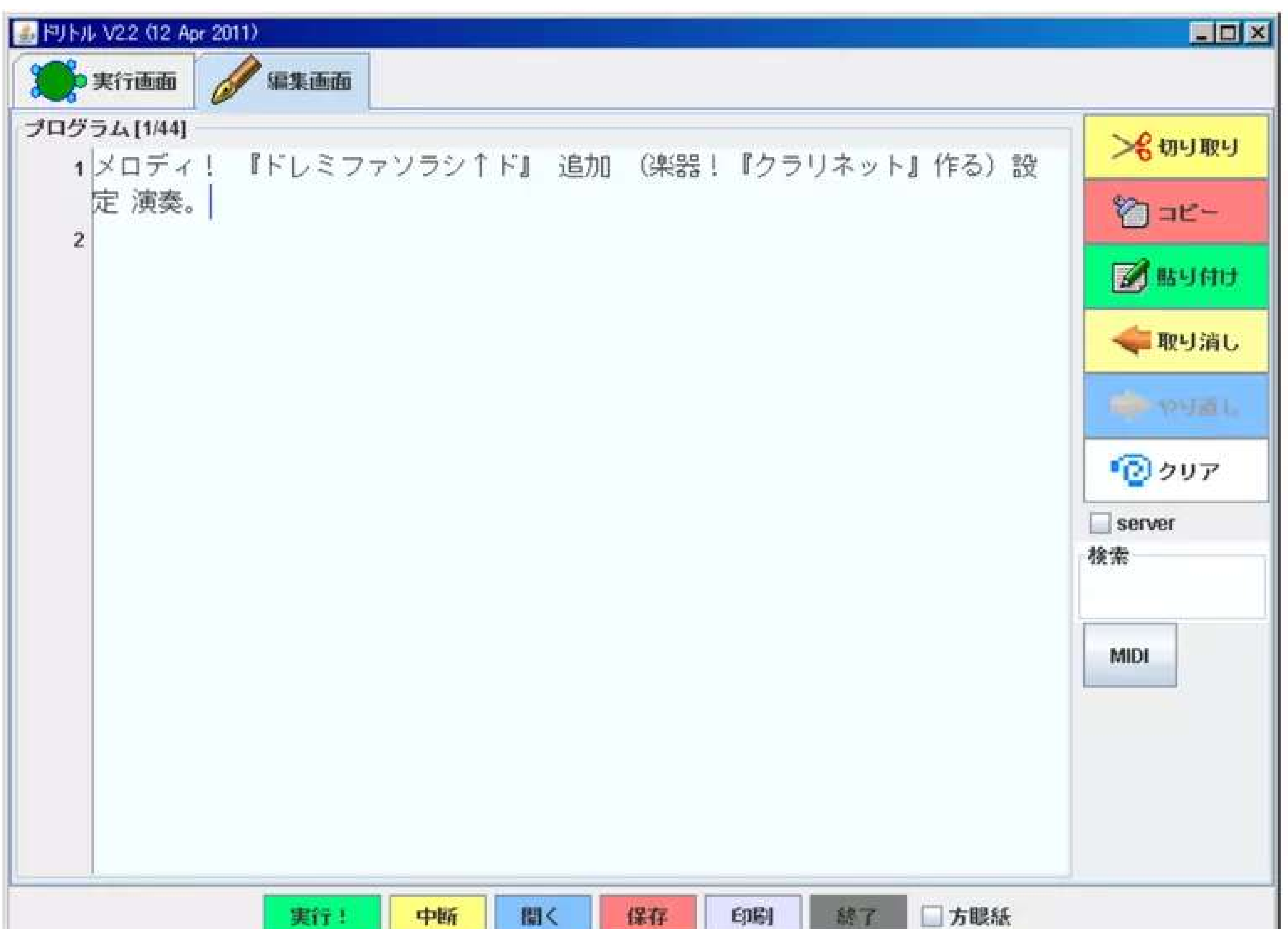
GM音源

- Javaが持っているMIDI音源やWindowsの内蔵MIDI音源はGM仕様
- GM仕様の音源は標準で128種類の音色を使うことができる
- 128種類の音色にはピアノの他、オーケストラで使う楽器が含まれてる
- ドリトルでにもこの128種類の音色を使うためのメソッドがある

0: Piano 1	26: Jazz Guitar	52: Choir Aahs	78: Whistle	104: Sitar
1: Piano 2	27: Clean Guitar	53: Voice Oohs	79: Ocarina	105: Banjo
2: Piano 3	28: Guitar Mutes	54: Synth Vox	80: Square Wave	106: Shamisen
3: Honky-tonk	29: Overdrive Guitar	55: Orchestra Hit	81: Saw Wave	107: Koto
4: E.Piano 1	30: DistortionGuitar	56: Trumpet	82: Synth Calliope	108: Kalimba
5: E.Piano 2	31: Guitar Harmonics	57: Trombone	83: Chiffer Lead	109: Bagpipe
6: Harpsichord	32: Acoustic Bass	58: Tuba	84: Charang	110: Fiddle
7: Clavinet	33: Fingered Bass	59: Mute Trumpet	85: Solo Vox	111: Shenai
8: Celesta	34: Picked Bass	60: French Horns	86: 5th Saw Wave	112: Tinker Bell
9: Glockenspiel	35: Fretless Bass	61: Brass	87: Bass _Lead	113: Agogo
10: Music Box	36: Slap Bass 1	62: Synth Brass 1	88: Fantasia	114: Steel Drum
11: Vibraphone	37: Slap Bass 2	63: Synth Brass 2	89: Warm Pad	115: Wood Block
12: Marimba	38: Synth Bass 1	64: Soprano Sax	90: Poly Synth	116: Taiko Drum
13: Xylophone	39: Synth Bass 2	65: Alto Sax	91: Space Voice	117: Melodic Tom
14: Tubular Bells	40: Violin	66: Tenor Sax	92: Bowed Glass	118: Synth Drum
15: Dulcimer	41: Viola	67: Baritone Sax	93: Metal Pad	119: Reverse Cymbal
16: Organ 1	42: Cello	68: Oboe	94: Halo Pad	120: Fret Noise
17: Organ 2	43: Contra Bass	69: English Horn	95: Sweep Pad	121: Breath Noise
18: Organ 3	44: Tremolo Strings	70: Bassoon	96: Ice Rain	122: Seashore
19: Church Organ	45: Pizzicato	71: Clarinet	97: Soundtrack	123: Bird
20: Reed Organ	46: Harp	72: Piccolo	98: Crystal	124: Telephone
21: Accordion	47: Timpani	73: Flute	99: Atmosphere	125: Helicopter
22: Harmonica	48: Strings	74: Recorder	100: Brightness	126: Applause
23: Bandoneon	49: Slow Strings	75: Pan Flute	101: Goblin	127: Gun Shot
24: Nylon Guitar	50: Synth Strings 1	76: Bottle Chiff	102: Echo Drops	
25: Steel Guitar	51: Synth Strings 2	77: Shakuhachi	103: Star Theme	

演奏する楽器の指定

- 楽器は楽器オブジェクトで作る
- 楽器の指定は番号か名前
  - 楽器！『クラリネット』作る。
- メロディーオブジェクトに楽器を設定すると設定した楽器で演奏する
- 演奏する楽器を指定しないとピアノで演奏
- 『ドレミファソラシ↑ド』をクラリネットで演奏



番号	名前	番号	名前	番号	名前
1	グランドピアノ	44	コントラバス	87	フィフスリード
2	ブライトピアノ	45	トレモロストリングス	88	ベースアンドリード
3	エレクトリックグランドピアノ	46	ピチカートストリングス	89	ニューエイジパッド
4	ホンキートンクピアノ	47	オーケストラハープ	90	ワームパッド
5	エレクトリックピアノ 1	48	ティンパニ	91	ポリシンセパッド
6	エレクトリックピアノ 2	49	ストリングス	92	クワイアパッド
7	ハープシコード	50	スローストリングス	93	ボウドパッド
8	クラビネット	51	シンセストリングス 1	94	メタリックパッド
9	チェレスタ	52	シンセストリングス 2	95	ハロパッド
10	グロッケンシュピール	53	コーラス	96	スウィープパッド
11	ミュージックボックス	54	ボイス	97	アイスレイン
12	ビブラフォン	55	シンセボイス	98	サウンドトラック
13	マリンバ	56	オーケストラヒット	99	クリスタル
14	シロフォン	57	トランペット	100	アトモスフィア
15	チューブラーベル	58	トロンボーン	101	ブライトネス
16	ダルシマー	59	チューバ	102	ゴブリン
17	ドローバーオルガン	60	ミュートトランペット	103	エコードロップ
18	パーカッシブオルガン	61	フレンチホルン	104	エスエフ
19	ロックオルガン	62	ブラスセクション	105	シタール
20	チャーチオルガン	63	シンセブラス 1	106	バンジョー
21	リードオルガン	64	シンセブラス 2	107	三味線
22	アコーディオン	65	ソプラノサックス	108	琴
23	ハーモニカ	66	アルトサックス	109	カリンバ
24	タンゴアコーディオン	67	テナーサックス	110	バグパイプ
25	ナイロンギター	68	バリトンサックス	111	フィドル
26	スティールギター	69	オーボエ	112	シャナイ
27	ジャズギター	70	イングリッシュホルン	113	ティンクルベル
28	クリーンギター	71	バスーン	114	アゴゴ
29	ミュートギター	72	クラリネット	115	スティールドラム
30	オーバードライブギター	73	ピッコロ	116	ウッドブロック
31	ディストーションギター	74	フルート	117	太鼓
32	ギターハーモニクス	75	リコーダー	118	メロディックタム
33	アコースティックベース	76	パンフルート	119	シンセドラム
34	フィンガーベース	77	ブロウボトル	120	リバースシンバル
35	ピックベース	78	尺八	121	ギターフレットノイズ
36	フレットレスベース	79	ホイッスル	122	ブレスノイズ
37	スラップベース 1	80	オカリナ	123	シーショア
38	スラップベース 2	81	スクウェアリード	124	バード
39	シンセベース 1	82	ソートゥースリード	125	テレフォン
40	シンセベース 2	83	カリオペ	126	ヘリコプター
41	バイオリン	84	チフリード	127	アプローズ
42	ビオラ	85	チャランゴ	128	ガンショット
43	チェロ	86	ボイスリード		

ドリトルのMML

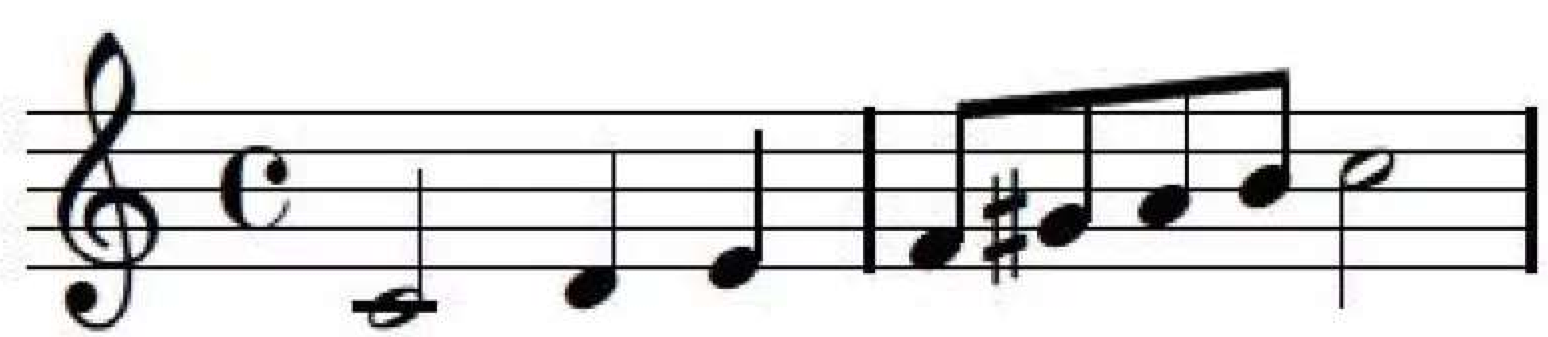
- ドリトルでは演奏する音楽をドリトル流のMMLで記述
- MMLは楽譜と同じように音符を演奏する順番に記述
- 音符は音符の高さと長さをセットで指定
  - [例] ド4・・・・中央ドの４部音符
- 長さを省略すると4部音符
- MMLはわかりやすい日本語で記述できる
- 英数字での記述も可能
- 音符と音符の間に空白は必要ない（あった方が見やすい）

MMLの記述方法（日本語表記）

- 音の高さ（音名で指定）
  - ドレミファソラシ　どれみふぁふそらし
- 音の長さ（n部音符のnの数字で指定）
  - １２４８16
- 付点音符
  - ．前の音をその半分の長さだけ伸ばす
- タイ
  - ＆ 音符をつなぎます　[例] ４＆8=付点4分音符
- 半音上下
  - ＃・・・半音上げる　♭・・・・半音下げる
- オクターブ上下
  - ↑・・・・1オクターブ上げる　↓1オクターブ下げる
- 休符
  - ・　[例] ・4 ・・・・ 4分休符
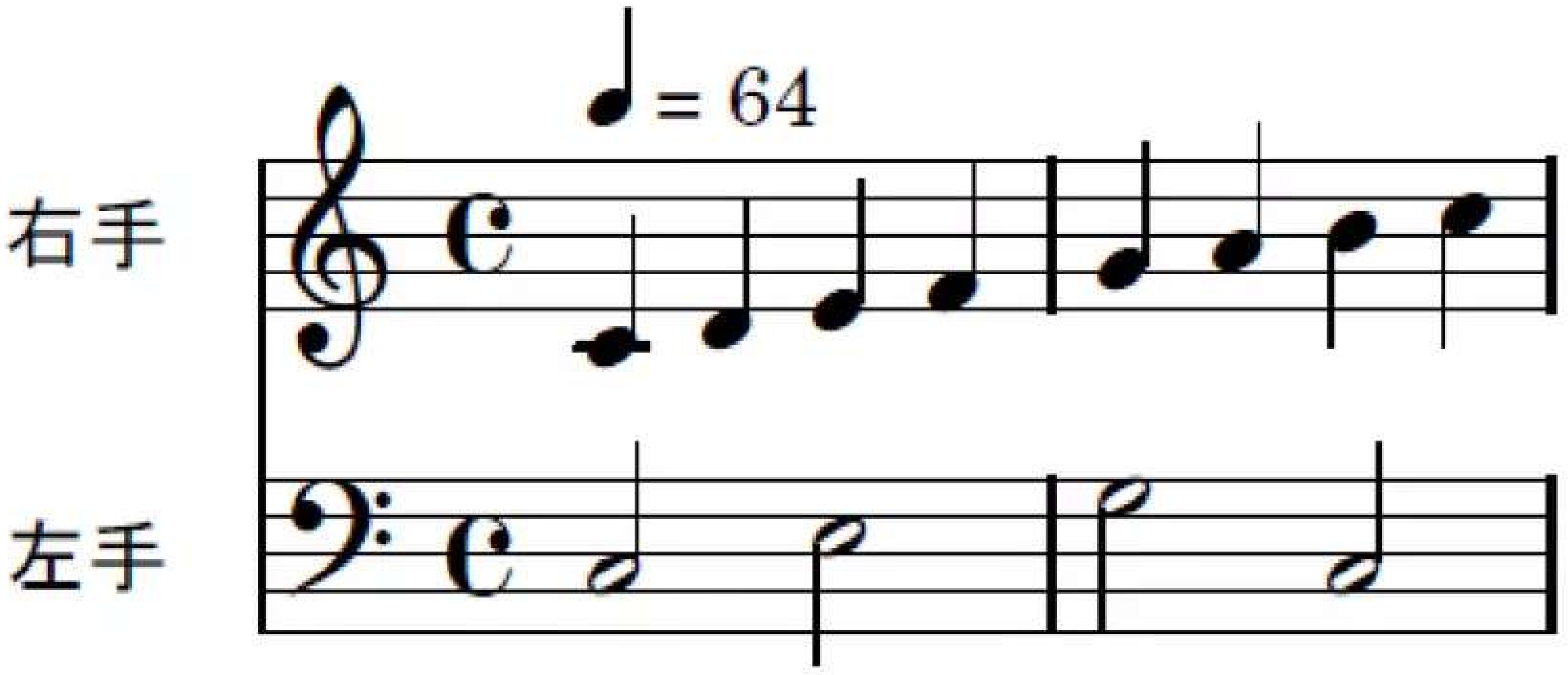
MMLの記述方法(英数字表記)

- 音の高さ(大文字小文字は区別されない)
  - c d e f g a b
- 音の長さ(n部音符のnの数字で指定)
  - 1 2 4 8 16
- 付点音符
  - . 前の音をその半分の長さだけ伸ばす
- タイ
  - & 音符をつなぎます [例] ４&8=付点4分音符
- 半音上下
  - #･･･半音上げる %････半音下げる
- オクターブ上下
  - ^ ･･･・ 1オクターブ上げる _ ･･･・ 1オクターブ



複数パートの演奏

- 複数のパートを同時に演奏するにはバンドオブジェクトを使う
  - バンド！（パート1）追加 （パート2）追加 ・・・（パートn）追加　演奏。
- パートはメロディオブジェクトを使って作る



バンド！（メロディ！作る『ドレミファソラシ↑ド』追加）追加 （メロディ！作る『↓ド2ミ2ソ2ド2 』追加）追加　演奏。

変数

- パートのメロディを変数(プロパティ)にする
- 変数の定義方法
  - オブジェクト名:プロパティ名＝値。
- オブジェクト名:　は省略できる
- ＝の右側には値を返すものを記述
  - 名前＝計算式。
  - 名前＝オブジェクト！メソッド。
  - 名前＝「ブロック」！実行。

ブロック

- 「文字列」のように書くと文字列の内容を実行できる
- この機能を使って関数（メソッド）を定義できる



ソフトシンセを使う

- ドリトルは外部MIDI音源を接続できる
  - GM仕様の音源
  - シンセサイザー
  - 電子ピアノ
  - ソフトシンセ（PCアプリ）





- GM音源として使えるソフトシンセ
  - SFZ Sample Player
    - http://www.cakewalk.com/Download/sfz.aspx
    - SFZ にはピアノ音源が付属しています。
    - SFZ を使うと演奏する音が格段に向上します。



- SAVIHost を使うとSFZをVSTiとして使える
  - SAVIHost
    - http://www.hermannseib.com/english/savihost.htm

音源を用意する

- SFZ はデジタル化された音を音源として使う
- SoundFont を入れ替えることで様々な楽器に変身
- GM仕様のSoundFont
  - GeneralUser GS（SoftSynth version）
  - http://www.schristiancollins.com/generaluser.php



MIDI信号の接続

- ドリトルのMIDI出力をソフトシンセに入力する
- MIDI信号の接続ソフト
  - MIDI Yoke Junction
    - http://www.midiox.com/
- ドリトルの出力とSFZの入力を同じにする





Ⅱ　演奏する音楽を作曲する

作曲の意味を考える

- メロディ！『ドレミファソラシ↑ド』 追加　演奏。
- MMLの部分を変数にしてみる
- MML＝『ドレミファソラシ↑ド』。メロディ！(MML) 追加　演奏。
- 変数をパラメータにする時は()で囲む
- ()の中は中置記法で記述できる
- 自動作曲とはMMLをプログラムで自動的に作るということ
- MMLは文字列なので文字列の処理が重要
- 配列を使うと楽譜を音符に分解できる

文字列オブジェクト

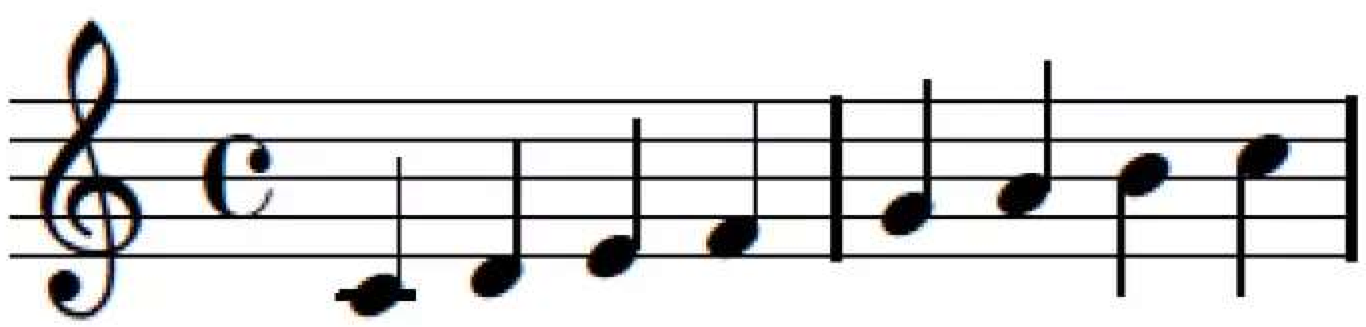
- 文字列オブジェクトは文字の並びを扱うオブジェクト
- 文字列オブジェクトのメソッド
    - +:　文字列を連結
    - 連結:　文字列を連結
    - 部分:　文字列を切り出す
    - 長さ？:　文字数を返す
    - 何文字目？:　文字列が何文字目に含まれるかを調べる
    - 含む？:　文字列を含むかを判定する
    - 分割:　区切り文字列を指定して分割した文字列が入った配列を返す
    - 置き換える:　文字列の一部を置き換えた文字列を返す
    - 全部置き換える: 文字列の一部を置き換えた文字列を返す
    - 実行:　文字列をドリトルのプログラムとみなして実行する

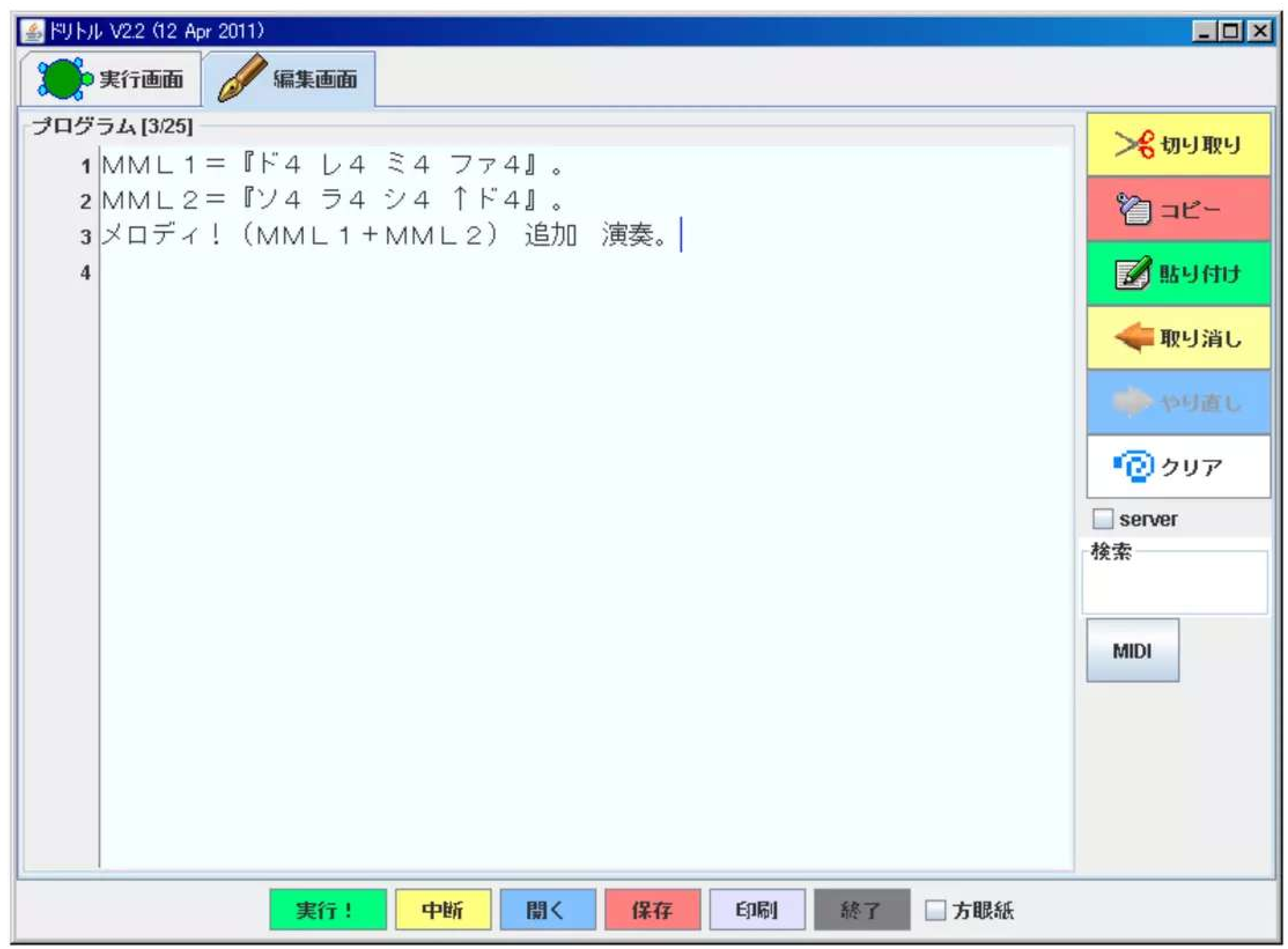
配列オブジェクト

- 配列オブジェクトは複数のデータを入れられるオブジェクト（要素の番号は1 から始まる）
- 配列の中に異なる種類のオブジェクトを入れることができる
- 配列オブジェクトのメソッド
  - 作る: 自分を複製して新しい配列を作る
  - 書く: 配列にオブジェクトを追加する
  - 挿入: 配列にオブジェクトを入れる
  - 上書き: 配列のオブジェクトを上書き
  - 読む: 配列の要素を返す
  - ランダムに選ぶ: 配列の要素をランダムに返す
  - 要素数？: 配列の要素数を返す
  - 消す: 配列の要素を消す
  - 位置で消す: 配列の要素を消す
  - クリア: 配列の要素をすべて消す
  - それぞれ実行: 配列の要素の数だけブロックを繰り返して実行
  - 選ぶ: 配列の各要素に対してブロックを実行し結果が真の要素の配列を返す
  - 結合: 配列の各要素を結合してひとつの文字列を返す

音符(文字)を連結してMMLを作る

- 文字列の連結は文字列を ＋ でつないでいく
- １小節目と2小節目をつなげてみましょう

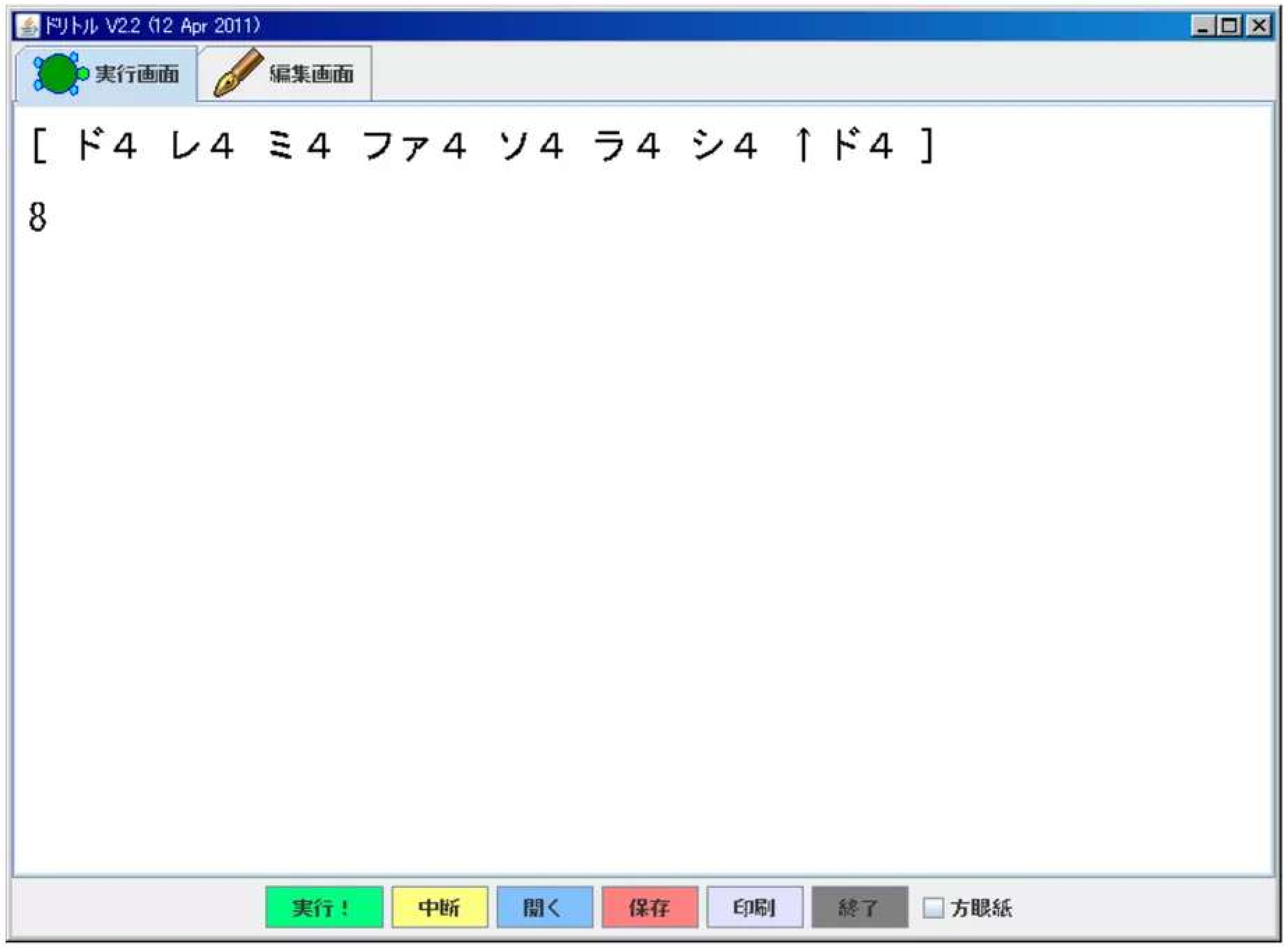




MMLを分割して音符を配列にする

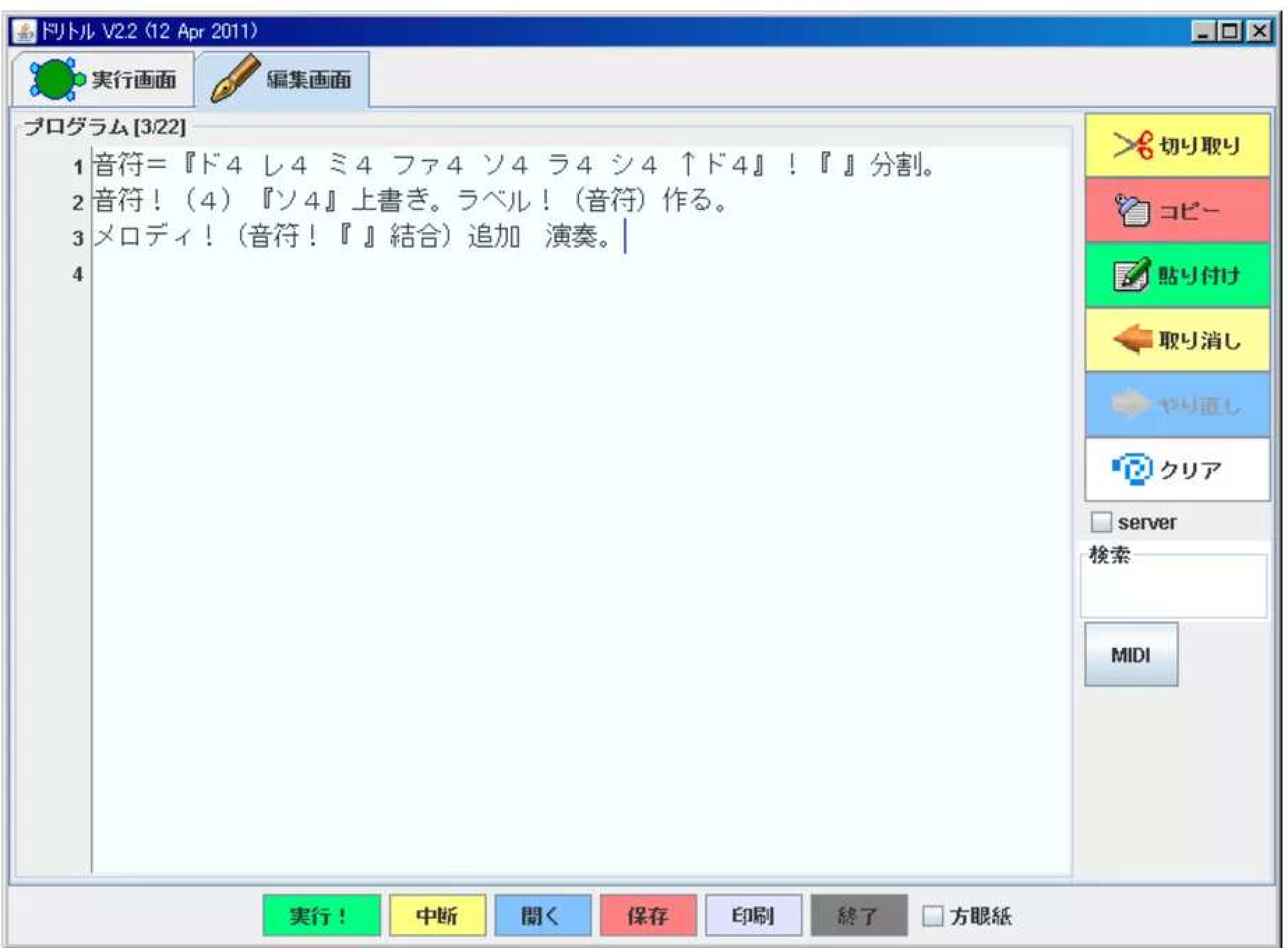
- 文字列オブジェクトの分割メソッドを使うと文字列を配列にできる
- ２小節の音符の並びをを一つ一つの音符に分解してみましょう
- 結果を確かめるにはラベルオブジェクトを使います

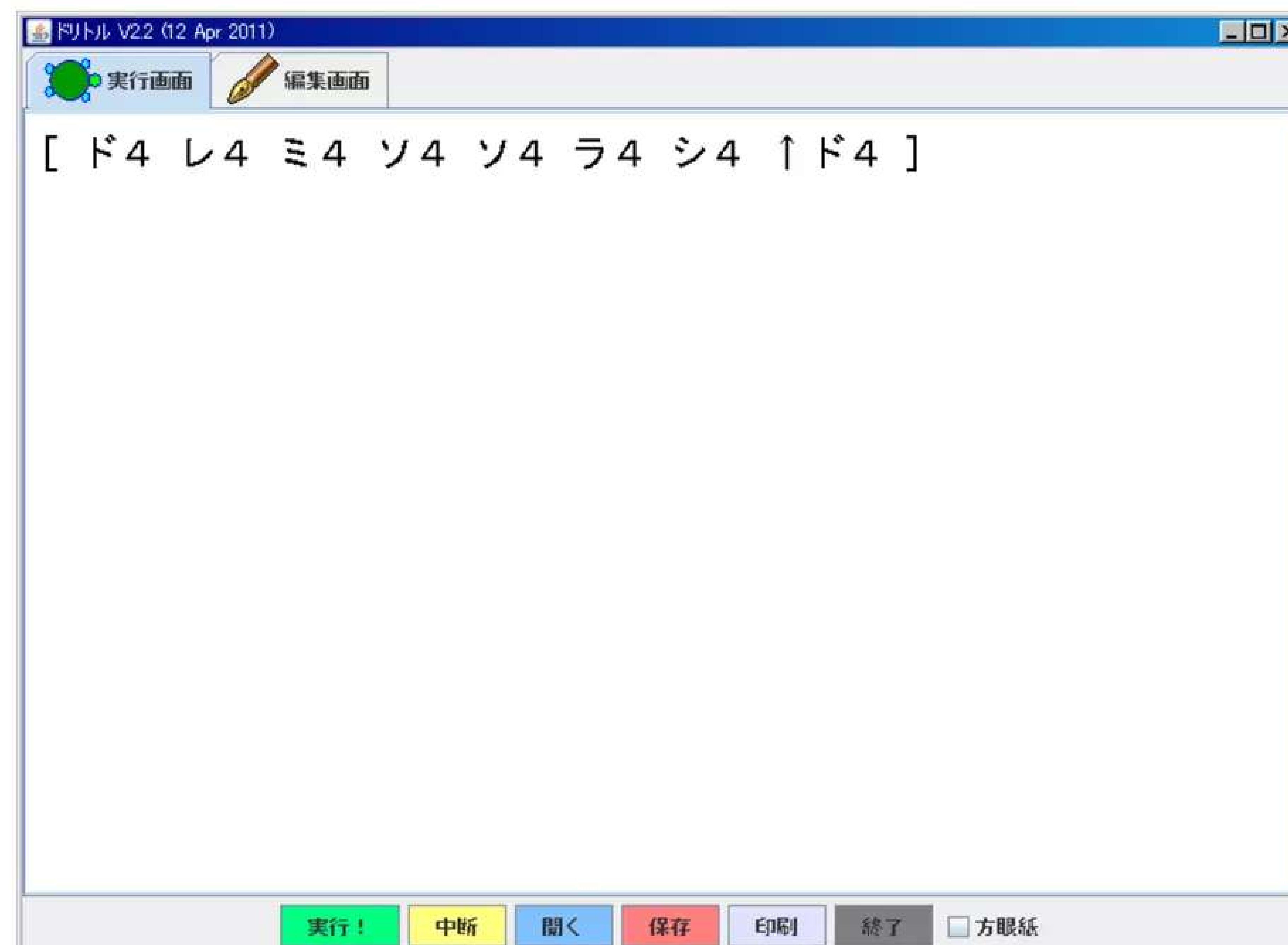




音符の配列を連結してMMLを作る

- 配列オブジェクトの上書きメソッドを使うと配列の内容を変更できる
- 配列の4番目のファの音をソに変更してみましょう
- 結合きメソッドを使うと配列の要素を結合して文字列にできる
- 配列の内容が変更されているか演奏させて確認してみましょう





音符の配列を逆向きにして連結

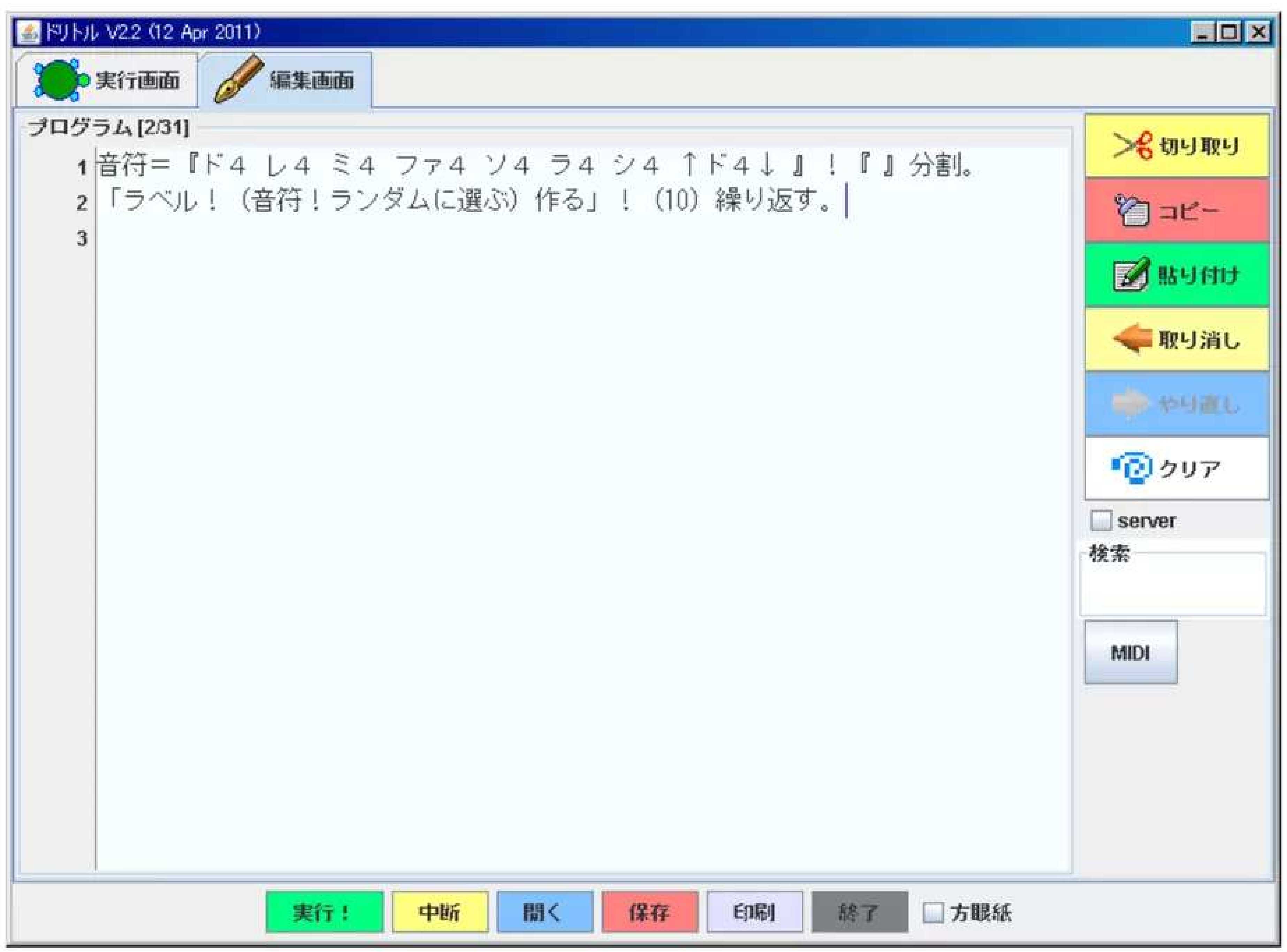
- 順番を指定して配列の要素を取り出す
  - 配列！（順番） 読む
- 繰り返しの構文
  - 「｜カウンタ変数｜ 処理」！（回数） 繰り返す
- オブジェクトの内容を表示
  - ラベル！（オブジェクト） 作る

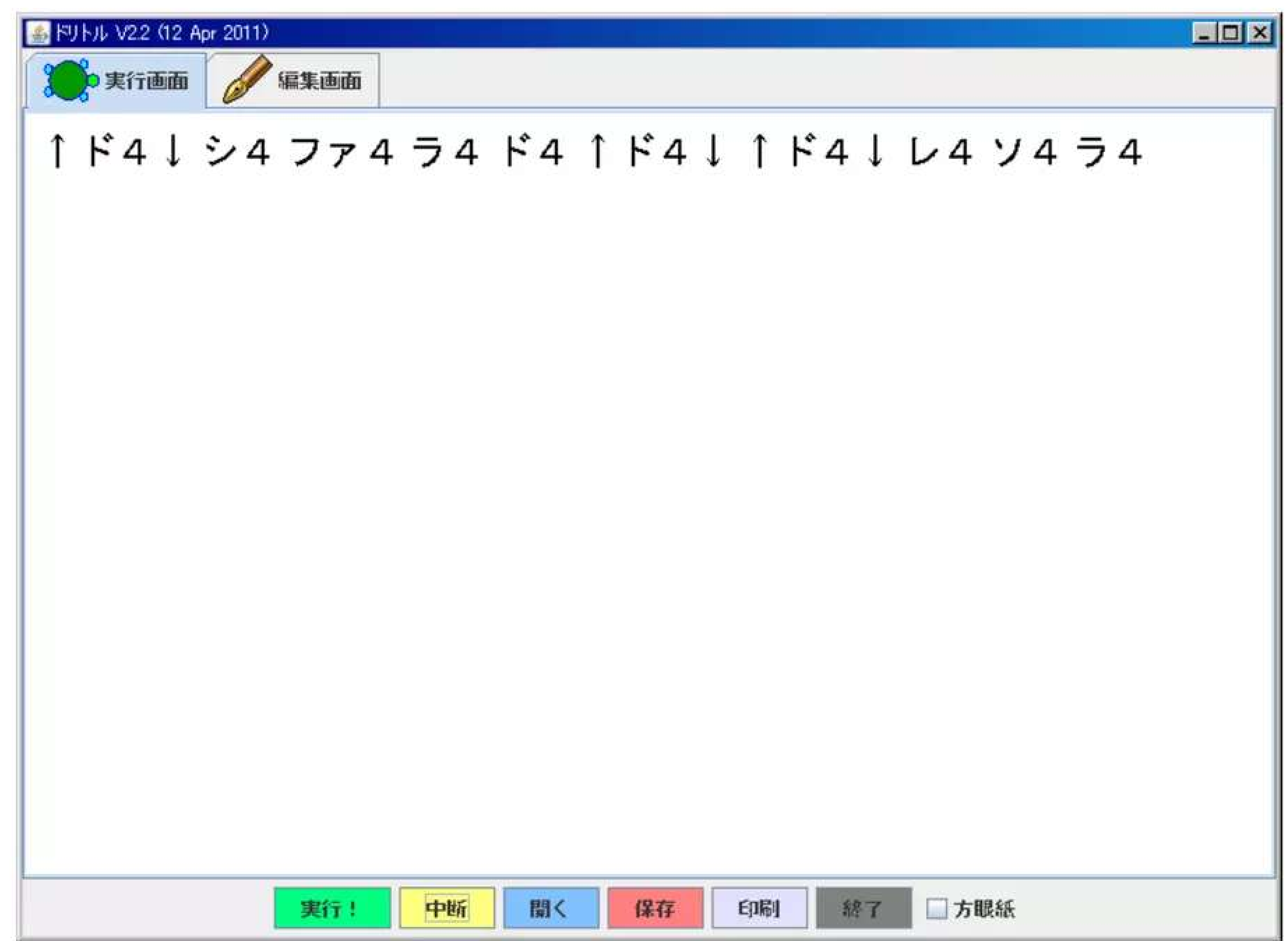




配列から音符をランダムに選択

- 配列の要素をランダムに選ぶ
  - 配列！ランダムに選ぶ
- 音階の配列から10個の音をランダムに選んでみましょう

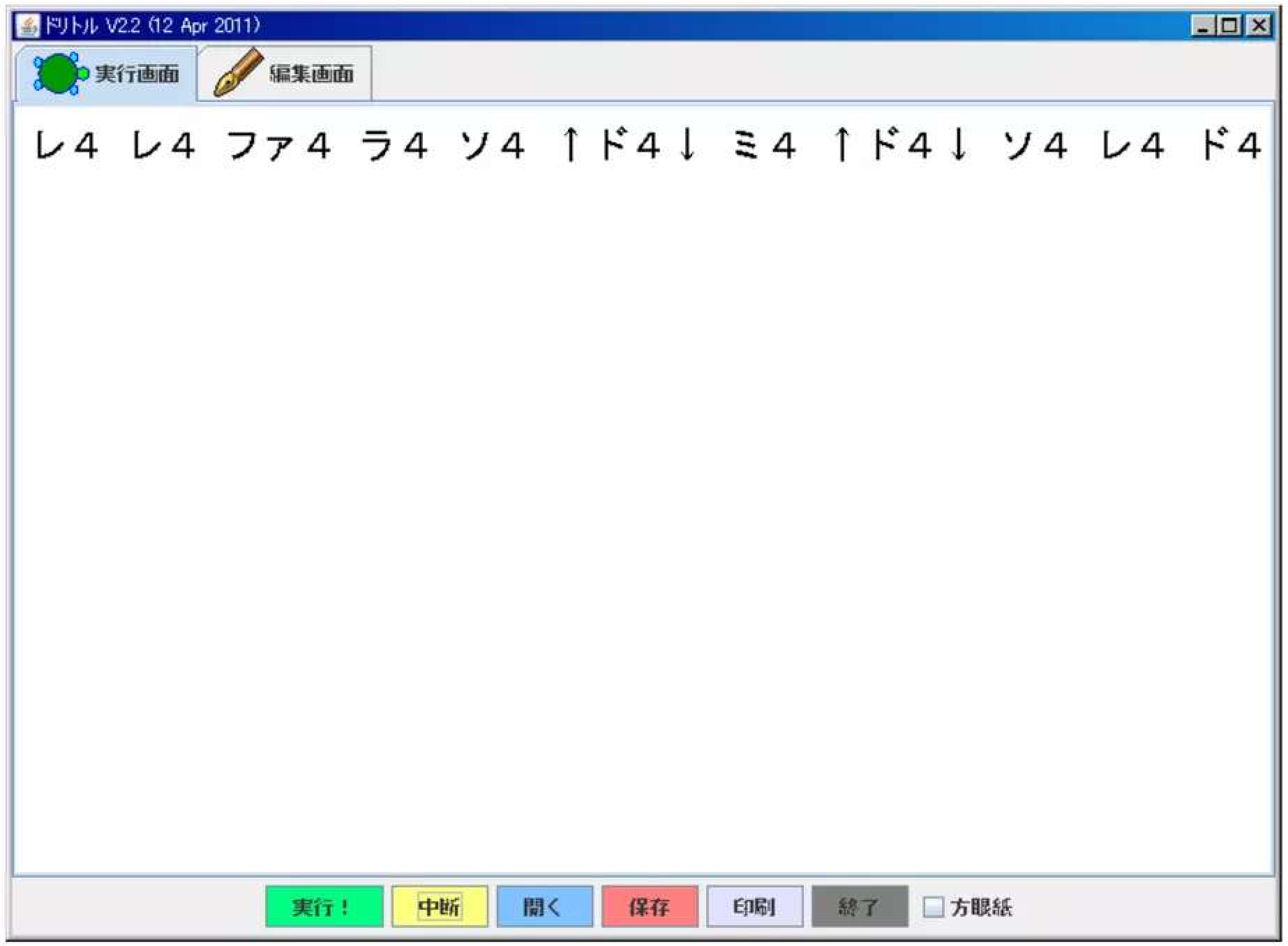




ランダムに選んだ音符を連結

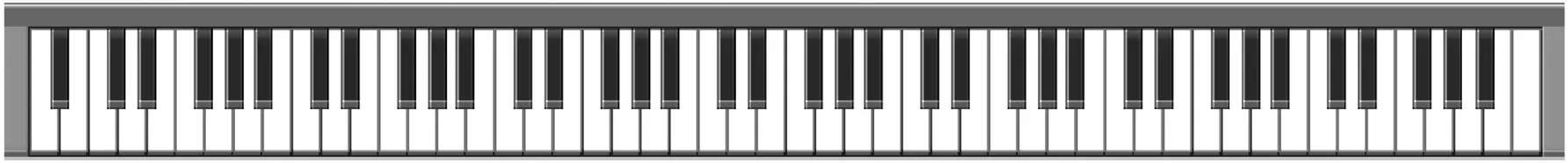
- 配列からランダムに音を選び4小節分のメロディーを作ってみます
- 音符の数は4×4小節数になります
- 表示をわかりやすくするために音符の間に空白を入れてみます
- 実行ボタンを押すたびに別のメロディーが自動的に作曲されます





MIDIノート番号

- MIDIの規格では 0～127 の番号で半音単位に C-2～G8 までの音程を表す
- 88鍵ピアノの音程はノートナンバー 21～108 に対応
- ドリトルの『ド』の音はC4でノート番号は60
- ノート番号の0はドリトルの『↓↓↓↓↓ド』



ノート番号を使う

- ノート番号を指定して音を指定する方法を考えてみましょう
- メロディオブジェクトの音上げるメソッドを使えばできます
- ドリトルの『ド』の音はノート番号60なので60上げてみましょう
- 『ド』の音が出ましたか？
- ソフトシンセを使えば鍵盤で確認できます

メロディ！作る『↓↓↓↓↓ド』追加（60）音上げる 演奏。



ピアノの鍵盤を演奏

- 88鍵ピアノの鍵盤はノートナンバー 21～108
- 88鍵全ての鍵盤の音を出してみましょう
- 繰り返しメソッドとカウンタを使えば簡単に実現できます
- 16部音符で 低→高→低 のように演奏させてみます

```
鍵盤＝メロディ！作る『↓↓↓↓↓ド16』追加。
「｜n｜メロディ！（鍵盤！（21-1 +n）音上げる）追加」！（88）繰り返す。
メロディ！ 演奏。// 低→高
「｜n｜メロディ！（鍵盤！（108+1-n）音上げる）追加」！（88）繰り返す。
メロディ！ 演奏。// 高→低
```



処理をブロックにまとめる

- 処理をブロックにまとめるとプログラムがわかりやすくなる
- ブロックには処理の内容がわかるような名前を付ける
- ブロックに付けた名前はメソッド名
  - ！名前　で呼び出す

```
処理の名前１＝「・・・・・・・・」。

処理の名前２＝「
  ・・・・・・
  ・・・・・・
」。
```

```
！処理の名前１。
！処理の名前２。
```

処理ブロックを定義してみる

- プログラムでは最初に実行する処理がある
  - 定数の設定や変数の初期化など
- 88鍵のキーボードの仕様を定義してみる
  - こうすることで様々な大きさのキーボードを定義できる

```
鍵盤を準備する＝「 // MIDIノート=0の音と鍵盤仕様の設定
  鍵盤＝メロディ！作る『↓↓↓↓↓ド16』追加。
  鍵数=88。左端=21。右端=108。
」。
```

- 演奏する処理もまとめてみましょう

```
左から右に弾く＝「  // 低→高
 「｜n｜メロディ！（鍵盤！（左端-1+n）音上げる）追加」！
（鍵数）繰り返す。
  メロディ！ 演奏。
」。
右から左に弾く＝「  // 高→低
 「｜n｜メロディ！（鍵盤！（右端+1-n）音上げる）追加」！
（鍵数）繰り返す。
  メロディ！ 演奏。
」。
```

ドリトルで構造化

- ブロックを順次に実行するように書いてみる
- 左から右に実行
  - ！鍵盤を準備する。！左から右に弾く。！右から左に弾く。
- 構造化することでわかりやすいプログラムになる

```
// ドリトルの構造化

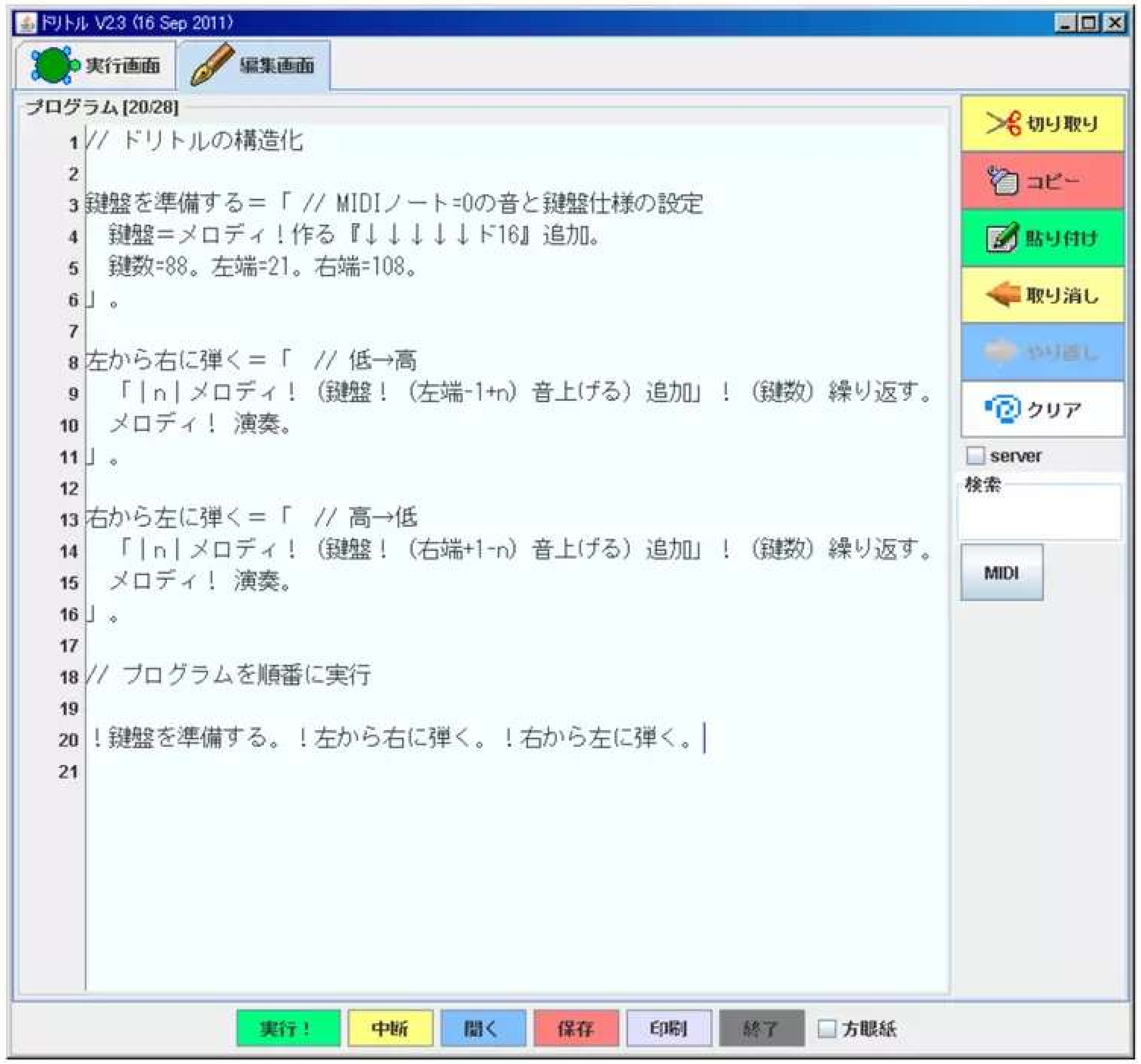
鍵盤を準備する＝「 // MIDIノート=0の音と鍵盤仕様の設定
  鍵盤＝メロディ！作る『↓↓↓↓↓ド16』追加。
  鍵数=88。左端=21。右端=108。
」。

左から右に弾く＝「  // 低→高
 「｜n｜メロディ！（鍵盤！（左端-1+n）音上げる）追加」！（鍵数）繰り返す。
  メロディ！ 演奏。
」。

右から左に弾く＝「  // 高→低
 「｜n｜メロディ！（鍵盤！（右端+1-n）音上げる）追加」！（鍵数）繰り返す。
  メロディ！ 演奏。
」。

// プログラムを順番に実行

！鍵盤を準備する。！左から右に弾く。！右から左に弾く。
```



多重ループ

- ブロックはネストできる
- 繰り返し（ループ）を２重に使ってみる
- 鍵盤を　低→高　２回弾いてみましょう

```
鍵盤＝メロディ！作る『↓↓↓↓↓ド16』追加。
// 低→高 2回
メロディ！(「｜i｜「｜n｜メロディ！（鍵盤！（20+n）音上げる）追加」！（88）繰り返す」！（2）繰り返す ) 演奏。
```

条件文

- 今度は　低→高　高→低　のように弾く処理を２重ループで作ってみる
- １回目と2回目で処理が異なるので条件により処理を変える必要がある
- ドリトルの条件文
  - 「…」！ なら「…」実行。
  - 「…」！ なら「…」そうでなければ「…」実行。

```
鍵盤＝メロディ！作る『↓↓↓↓↓ド16』追加。
// 低→高　高→低
メロディ！(「｜i｜「｜n｜「i==1」!なら「m=20+n」そうでなければ「m=109-n」実行。メロディ！（鍵盤！（m）音上げる）追加」！（88）繰り返す」！（2）繰り返す ) 演奏。
```

```
鍵盤＝メロディ！作る『↓↓↓↓↓ド16』追加。
// 低→高　高→低
メロディ！(「｜i｜「｜n｜「i==1」!なら「m=20+n」実行。「i==2」!なら「m=109-n」実行。メロディ！（鍵盤！（m）音上げる）追加」！（88）繰り返す」！（2）繰り返す ) 演奏。
```

インデント

- 横に長い文は読みにくい場合がある
- ドリトルは改行を自由にいれることができる
- 論理構造がわかりやすいように字下げをすること
- 繰り返しのパラメータに『回』を使ってわかりやすくできる
- 「‥‥」と[‥‥]　『‥‥』と"‥‥"　は同じ
- 『。』と『.』も同じ　大文字と小文字、全角と半角は区別されない

```
鍵盤＝メロディ！作る "↓↓↓↓↓ド16" 追加.
// 低→高　高→低
メロディ！(
  [ | i |
    [ | n |
       [i==1]! なら [m=20 +n] 実行.
       [i==2]! なら [m=109-n] 実行.
      メロディ！(鍵盤！(m)音上げる)追加
    ]！ 88回 繰り返す
  ]！ 2回 繰り返す
 ) 演奏.
```



ノート番号を音名に変換

- 音をランダムに作るには文字（MML）より数字（ノート番号）の方が便利
- ノート番号をドリトルのMMLに変換するメソッド（関数）を作ってみる
  - 呼び出し方法　！(ノート番号) NoteName
- 関数の書き方
  - 関数名＝[|パラメータ並び; ローカル変数並び| 処理・・・・ 戻り値]。
- １オクターブは12音
  - オクターブはノート番号を12で割った商、音名は余りで求められる
- ドリトルの音名はオクターブの指定が相対指定
- オクターブ上下させた場合は、元に戻しておく
  - オクターブ指定＋音名＋オクターブ戻す

```
// MIDIノート番号からドリトルの音名を作る
NoteName=[|番号; 音階 前 後 octv|
  音階="ド:ド#:レ:レ#:ミ:ファ:ファ#:ソ:ソ#:ラ:ラ#:シ" ! ":" 分割.
  前="_____:____:___:__:_::^:^^:^^^:^^^^:^^^^^]"! ":" 分割.
  後="^^^^^:^^^^:^^^:^^:^::_:__:___:____:_____]"! ":" 分割.
  octv=floor(番号/12)+1.
  (前 ! (octv) 読む) + (音階 ! (番号%12+1) 読む) + (後 ! (octv) 読む).
].
```

音名をノート番号に変換

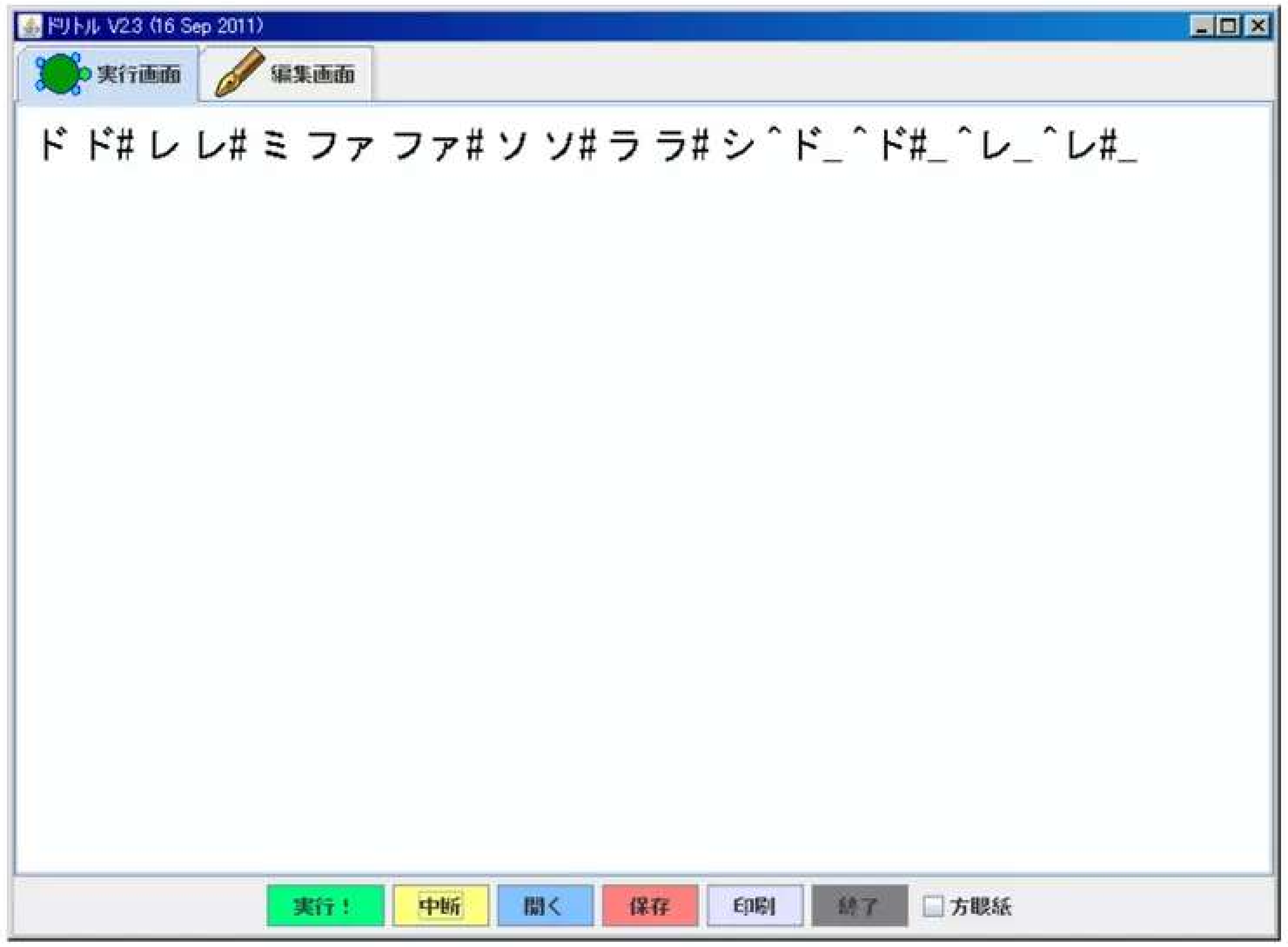
- ドリトルの音名をMIDIノート番号に変換する関数も作ってみましょう。
  - 呼び出し方法　！(音名) NoteNumber
- 音名（文字列）を１文字ずつ走査して計算します
  - オクターブ記号なら12倍プラスあるいはマイナス
  - 音階なら『ド』の音からの相対量をプラス
  - 半音記号は1音プラスあるいはマイナス

```
// ドリトルの音名からMIDIノート番号を作る
NoteNumber=[|音名; mns pls onk scl bis k|
  mns=0. pls=0. onk=0.
  scl="c,d,e,f,g,a,b,ド,レ,ミ,ファ,ソ,ラ,シ,#,%,＃,♭" ! "," 分割.
  bis="0,2,4,5,7,9,11,0,2,4,5,7,9,11,1,-1,1,-1" ! "," 分割.
  k=1.[
    [音名! (scl!(k)読む)含む?]! なら [onk=onk+(bis!(k)読む)]実行. k=k+1.
  ] ! (scl! 要素数?) 繰り返す.
  k=1.[
    [(音名！ (k)(1) 部分)! "↓" 含む?]! なら [mns=mns+1]実行.
    [(音名！ (k)(1) 部分)! "↑" 含む?]! なら [pls=pls+1]実行.
    k=k+1.
  ] !(音名! 長さ?)繰り返す.
  60 + onk + (pls*12) - (mns*12).
].
```

作曲用のライブラリーを作る

- 作ったた関数は一つのファイルにまとめてライブラリーに
  - compose.ini という名前で保存
- このソースを呼び出すには
  - システム！ "compose" 使う。
- 関数をテストするプログラムを作ってみましょう

```
// NoteNumber&NoteName テスト
システム！ "compose" 使う。
n=(! ("↓シ") NoteNumber).
「|i| ラベル！(! (n+i) NoteName)作る. 」! (16) 繰り返す。
```



ランダム関連の関数

- ドリトルの乱数 random(n) は 1 ～ n の間のランダムな数を返す
- 指定した範囲のランダムな数を返す関数を作ってみる
  - 呼び出し方法　！(最小値)(最大値) RandomNumber

```
// 指定範囲の乱数を作る
RandomNumber=[|r_min r_max|
  random(r_max - r_min) + r_min.
].
```

- ドリトルには配列から要素をランダムに選ぶメソッドがある、が、あえて
- カンマ区切りの要素の並び（文字列）から要素を選ぶ関数を作ってみる
  - 呼び出し方法　！(最小値)(最大値) RandomNumber
- ランダム関連の関数も「compose.ini」に入れておきましょう

```
// カンマ区切りのデータからランダムに選ぶ
RandomSelect=[|slc; ary|
  ary=slc ! "," 分割.  ary! (random(ary! 要素数?)) 読む.
].
```

音域指定でランダムな音を作る

- 『↓ソ』～『ミ↑↑』の範囲の音を100個ランダムに作って演奏させてみる
- ドリトルの空のMMLを作り、1音づつ音を作って追加していく
- これで12音技法の現代音楽ができあがり
- 実行！ボタンを押して演奏させてみましょう
- 音をランダムにしているので実行する度に別の曲が演奏される

```
初期処理＝「
  システム！ "compose" 使う。
」。
作曲処理＝「
  下限＝! ("↓ソ") NoteNumber。上限＝! ("ミ↑↑") NoteNumber。mml=""。
  「
    音符＝!（! (下限)(上限) RandomNumber）NoteName。
    mml＝mml＋音符。
  」！ (100) 繰り返す。
」。
演奏処理＝「
  メロディ！ (mml) 追加 演奏。
」。

！初期処理。！作曲処理。！演奏処理。
```

音符の長さもランダムにしてみる

- 音符の長さも可変にしてみます
- 2分音符、4分音符、8分音符の中からランダムに選ぶようにする
- さらに現代音楽作品っぽくなってきた
- 何度か実行！ボタンを押して偶然性が作り出す音楽を楽しんでみましょう

```
初期処理＝[
  システム！ "compose" 使う。
]。
作曲処理＝[
  下限＝! ("↓ソ") NoteNumber。上限＝! ("ミ↑↑") NoteNumber。
  mml=""。[
    高さ＝!（! (下限)(上限) RandomNumber）NoteName。
    長さ＝(!("2,4,8") RandomSelect)。
    音符＝高さ＋長さ。
    mml＝mml＋音符。
  ]！ (100) 繰り返す。
]。
演奏処理＝「
  メロディ！ (mml) 追加 演奏。
」。

！初期処理。！作曲処理。！演奏処理。
```

作曲処理の汎用化

- ランダムに音を作る処理を汎用化してみる。
- いわば「作曲メソッド」の作成
- パラメータ

  音符数・・・・作成する音符の数

  下限・・・・作成する一番下のの音

  上限・・・・作成する一番上のの音

  使う音符・・・・使う音符の長さ（長さを『,』で区切って並べる）
- 戻り値

  音符配列・・・・音符数分の音の高さの配列

  長さ配列・・・・音符数分の音の長さの配列
- 呼び出し方法

  ! (音符数)(下限)(上限)(使う音符)(音符配列)(長さ配列) compose1.

作曲メソッドを作る

- ランダムに音を作る「作曲メソッド」を作ってみる
- ランダム関連の関数を作ってあるので簡単につくることができる

```
compose1＝[|音符数 下限 上限 使う音符 音符配列 長さ配列|
  [|i|
    note_n=(!(!(!(下限)NoteNumber)(!(上限)NoteNumber)RandomNumber) NoteName).
    note_l=(!(使う音符) RandomSelect).
    音符配列! (i)(note_n) 挿入.
    長さ配列! (i)(note_l) 挿入.
  ] ! (音符数) 繰り返す.
].
```

音符数指定で作曲

- compose1 も copose.ini に入れておく
- それでは compose1 を使って作曲してみる
- 『↓ソ』〜『ミ↑↑』の範囲の音を100個作り演奏させてみる
- 音符の長さは、2分音符、4分音符、8分音符からランダムに選んでみる

```
初期処理＝[
  システム！ "compose" 使う.
].

作曲処理＝[
  音符数＝100. 下限＝"↓ソ". 上限＝"ミ↑↑". 使う音符＝"2,4,8".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  ! (音符数)(下限)(上限)(使う音符)(音符配列)(長さ配列) compose1.
  mml="". [|i|
    音符＝（音符配列！(i)見る）＋（長さ配列！(i)見る）.
    mml＝mml＋音符.
  ]！ (音符数) 繰り返す.
].

演奏処理＝[
  メロディ！ (mml) 追加  演奏.
].

！初期処理. ！作曲処理. ！演奏処理.
```

小節数を数える

- 音符の長さは様々なので・・・・。
- 小節数を指定するためには音符の長さの計算が必要
- 計算方法
  - 1小節の長さ=1　2分音符=0.5　4分音符=0.25　・・・・
  - n分音符の長さ=1/n
- 拍子（タイムベース）により小節数の長さは変わる
- m/n の小節数分の長さの計算
  - nの長さ=m/n*小節数
- 小節数の長さを超えるまで音符を生成
- 小節数を超えたところで小節数に合うように最後の音符の長さを調整

小節数指定の作曲メソッド

- 小節数を指定してランダムな音を作る作曲メソッドを作ってみる

  パラメータ

  小節数・・・・作成する音符の数

  拍子・・・・タイムベース

  下限・・・・作成する一番下のの音

  上限・・・・作成する一番上のの音

  使う音符・・・・使う音符の長さ（長さを『,』で区切って並べる）

  戻り値

  音符配列・・・・音符数分の音の高さの配列

  長さ配列・・・・音符数分の音の長さの配列

```
compose2＝[|小節数 拍子 下限 上限 使う音符 音符配列 長さ配列; m tt tm to w_base|
  w_base=拍子! "/" 分割.
  m=小節数 * (w_base! (1) 読む) * (4/(w_base! (2) 読む)).
  i=1. tt=0.
  [tt < m] ! の間 [
    note_n=(!(!(!(下限)NoteNumber)(!(上限)NoteNumber)RandomNumber)
NoteName).
    tm=(!(使う音符) RandomSelect).
    to=tt. tt=tt+(1/tm)*4.
    [tt > m]! なら [tm=1/(m-to)*4] 実行.
    note_l=tm.
    [(m-to)==0.75]! なら [note_l="8.". tt=m ] 実行.
    [(m-to)==1.50]! なら [note_l="4.". tt=m ] 実行.
    [(m-to)==1.75]! なら [note_l="4..". tt=m ] 実行.
    音符配列! (i)(note_n) 挿入.
    長さ配列! (i)(note_l) 挿入.
    i=i+1.
  ] 実行.
].
```

小節数指定で作曲

- compose2 も copose.ini に入れておく
- それでは compose2 を使って作曲してみる
- 『↓ソ』～『ミ↑↑』の範囲の音を16小節分作り演奏させてみる
- 生成されたMMLを表示してみる（1行で表示するので最初の部分だけを確認）

```
初期処理＝[
  システム！ "compose" 使う.
].

作曲処理＝[
  小節数＝16. 拍子＝"4/4". 下限＝"↓ソ". 上限＝"ミ↑↑". 使う音符="2,4,8".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  ! (小節数)(拍子)(下限)(上限)(使う音符)(音符配列)(長さ配列) compose2.
  音符数=(音符配列! 要素数?).
  mml=""。[|i|
    音符＝（音符配列！(i)見る)＋（長さ配列！(i)見る）.
    mml＝mml＋音符＋" ".
  ]！ (音符数) 繰り返す.
].
演奏処理＝[
  メロディ！ (mml) 追加　演奏. ラベル！ (mml) 作る.
].

 ！初期処理. ！作曲処理. ！演奏処理.
```



回文的なMMLを作る

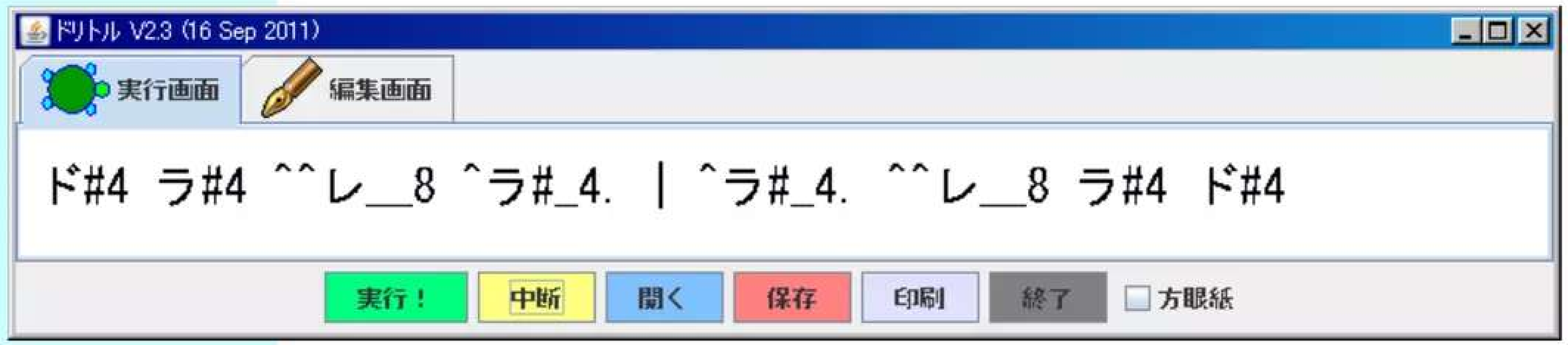
- 前から演奏しても後ろから演奏しても同じになる回文的な曲を作る
- 作曲でできた配列を逆に辿れば簡単に作ることができる
- ２小節分の曲を作って実験してみましょう

```
初期処理＝[
  システム！ "compose" 使う.
].

作曲処理＝[
  小節数＝2.拍子＝"4/4". 下限＝"↓ソ". 上限＝"ミ↑↑". 使う音符="2,4,8".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  !(小節数)(拍子)(下限)(上限)(使う音符)(音符配列)(長さ配列) compose2.
  音符数=(音符配列! 要素数?).
  mml_1="". [|i|
    音符＝(音符配列！(i)見る)＋(長さ配列！(i)見る).
    mml_1＝mml_1＋音符＋" ".
  ]！ (音符数) 繰り返す.
  mml_2="". [|i|
    音符＝(音符配列！(音符数+1-i)見る)＋(長さ配列！(音符数+1-i)見る).
    mml_2＝mml_2＋音符＋" ".
  ]！ (音符数) 繰り返す.
].

演奏処理＝[
  メロディ！ (mml_1 + mml_2) 追加  演奏.
  ラベル！ (mml_1 + "| " + mml_2) 作る.
].

！初期処理. ！作曲処理. ！演奏処理.
```



2パートのMMLを作る

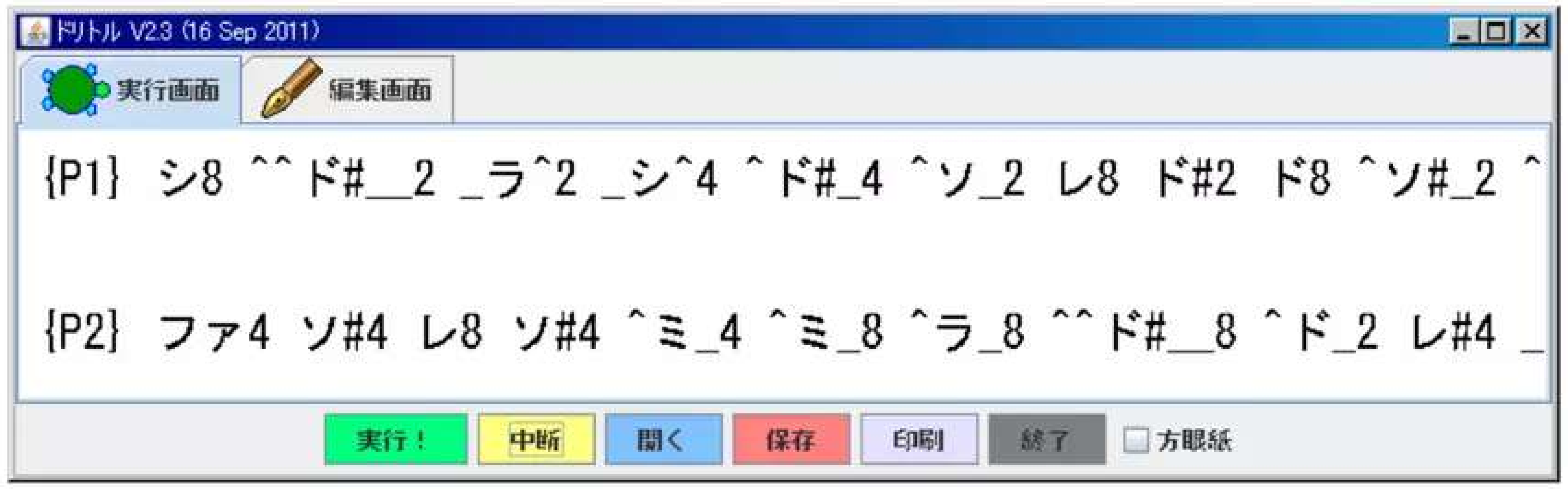
- 今度は配列を逆順にしたメロディーを別のパートにして2パートにしてみる
- 演奏処理の部分を変更するだけ！
- まずは8小節演奏してみましょう

```
初期処理＝[
  システム！ "compose" 使う.
].

作曲処理＝[
  小節数＝8. 拍子＝"4/4". 下限＝"↓ソ". 上限＝"ミ↑↑". 使う音符="2,4,8".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  ! (小節数)(拍子)(下限)(上限)(使う音符)(音符配列)(長さ配列) compose2.
  音符数=(音符配列! 要素数?).
  mml_1="". [|i|
    音符＝（音符配列！(i)見る）＋（長さ配列！(i)見る）.
    mml_1＝mml_1＋音符＋" ".
  ]！ (音符数) 繰り返す.
  mml_2="". [|i|
    音符＝（音符配列！(音符数+1-i)見る）＋（長さ配列！(音符数+1-i)見る）.
    mml_2＝mml_2＋音符＋" ".
  ]！ (音符数) 繰り返す.
].

演奏処理＝[
  バンド！（メロディ！作る (mml_1) 追加）追加（メロディ！作る (mml_2) 追
加）追加 演奏.
  ラベル！ ("{P1} "＋mml_1) 作る. ラベル！ ("{P2} "＋mml_2) 作る 次の行.
].

！初期処理. ！作曲処理. ！演奏処理.
```



1パートを2拍ずらす

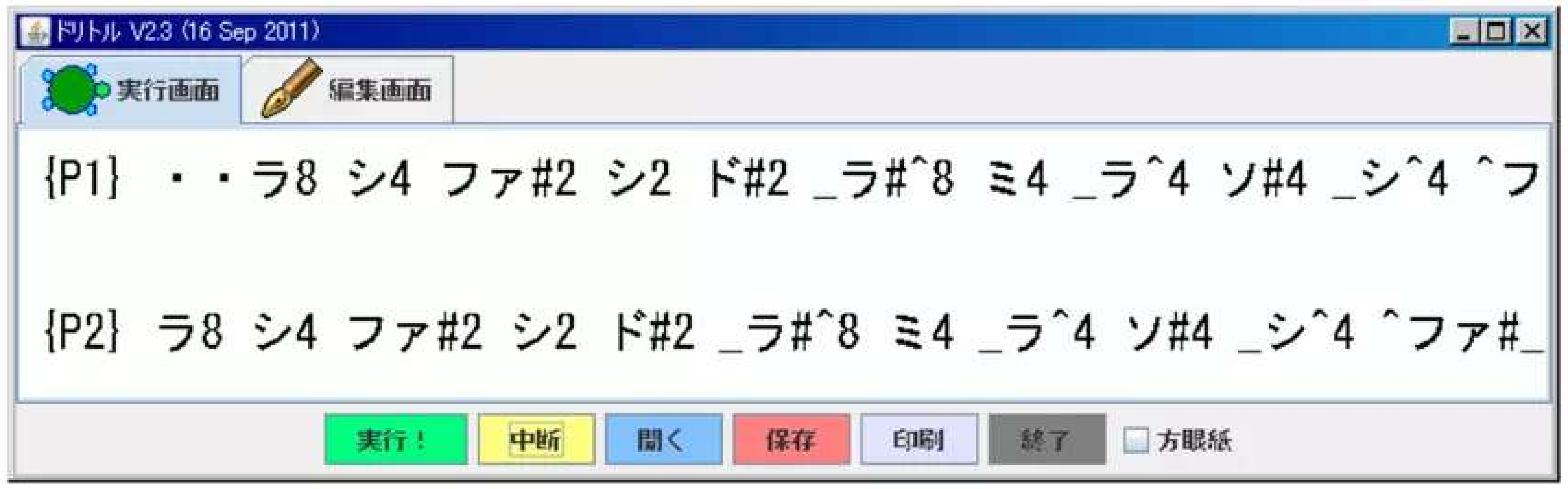
- 今度は最初のパートを2拍後ろにずらして（休符『・』を入れる）みる
- 16小節演奏させてみましょう
- 同じメロディをずらして重ねるだけで複雑なメロディができあがる

```
初期処理＝[
  システム！ "compose" 使う.
].

作曲処理＝[
  小節数＝16. 拍子＝"4/4". 下限＝"↓ソ". 上限＝"ミ↑↑". 使う音符="2,4,8".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  ! (小節数)(拍子)(下限)(上限)(使う音符)(音符配列)(長さ配列) compose2.
  音符数=(音符配列! 要素数?).
  mml_1="". [|i|
    音符＝（音符配列！(i)見る）＋（長さ配列！(i)見る）.
    mml_1＝mml_1＋音符＋" ".
  ]！ (音符数) 繰り返す.
  mml_2="". [|i|
    音符＝（音符配列！(音符数+1-i)見る）＋（長さ配列！(音符数+1-i)見る）.
    mml_2＝mml_2＋音符＋" ".
  ]！ (音符数) 繰り返す.
].

演奏処理＝[
  part_1＝"･･"＋mml_1＋mml_2. part_2＝mml_1＋mml_2＋"･･".
  バンド！（メロディ！作る (part_1) 追加）追加（メロディ！作る (part_2) 追加）追加 演奏.
  ラベル！ ("{P1} "＋part_1) 作る. ラベル！ ("{P2} "＋part_2) 作る 次の行.
].

！初期処理. ！作曲処理. ！演奏処理.
```



テンポをランダムにする

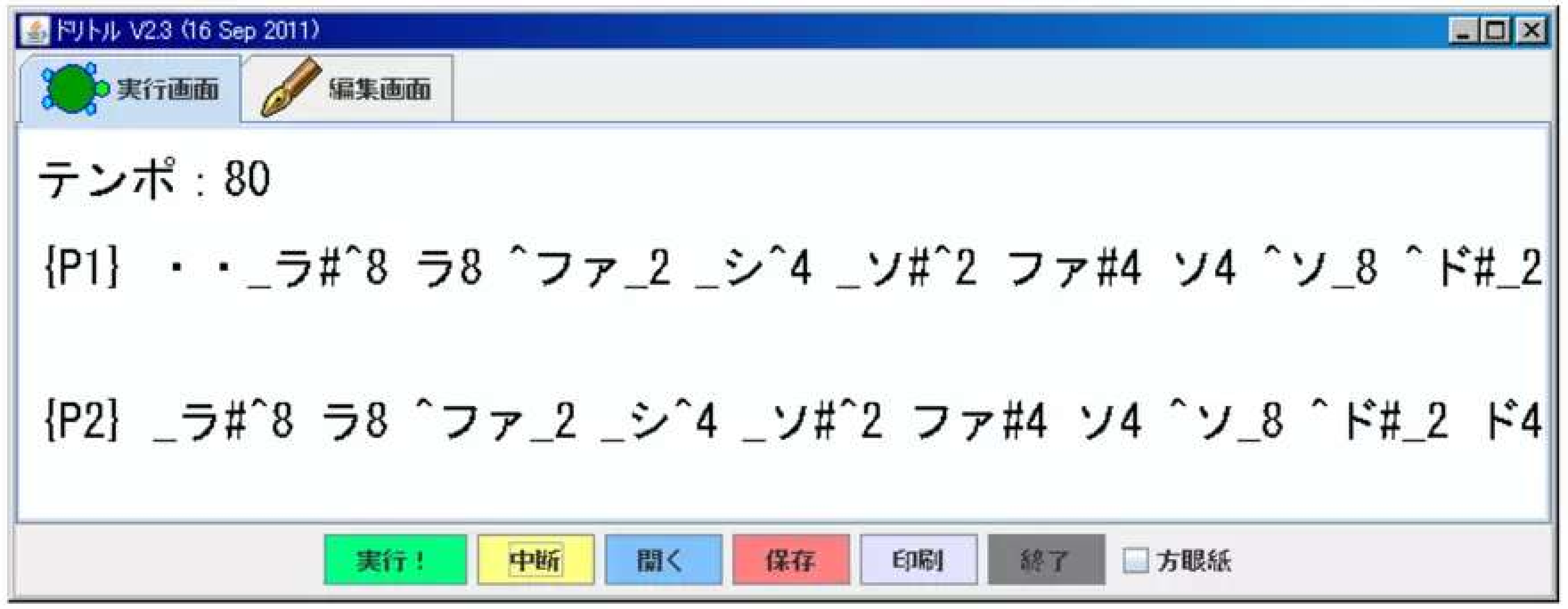
- 曲のテンポを変えると曲の雰囲気も変わる
- テンポをランダムに選ぶ処理を初期処理に加えてみる
- テンポはバンドオブジェクトで指定

```
初期処理＝[
  システム！ "compose"  使う.
  速さ＝(!("48,64,80,96,112,128") RandomSelect) .
].

作曲処理＝[
  小節数＝16. 拍子＝"4/4". 下限＝"↓ソ". 上限＝"ミ↑↑". 使う音符="2,4,8".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  ! (小節数)(拍子)(下限)(上限)(使う音符)(音符配列)(長さ配列) compose2.
  音符数=(音符配列! 要素数?).
  mml_1="". [|i|
    音符＝（音符配列！(i)見る）＋（長さ配列！(i)見る）.
    mml_1＝mml_1＋音符＋" ".
  ]！ (音符数) 繰り返す.
  mml_2="". [|i|
    音符＝（音符配列！(音符数+1-i)見る）＋（長さ配列！(音符数+1-i)見る）.
    mml_2＝mml_2＋音符＋" ".
  ]！ (音符数) 繰り返す.
].

演奏処理＝[
  part_1＝"･･"＋mml_1. part_2＝mml_1＋"･･".
  バンド！（メロディ！作る (part_1) 追加）追加（メロディ！作る (part_2) 追加）追加 （速
さ/2）テンポ 演奏.
  ラベル！ ("テンポ:"＋速さ) 作る.
  ラベル！ ("{P1} "＋part_1) 作る 次の行.
  ラベル！ ("{P2} "＋part_2) 作る 次の行.
].

！初期処理. ！作曲処理. ！演奏処理.
```



音符長セットをランダムにする

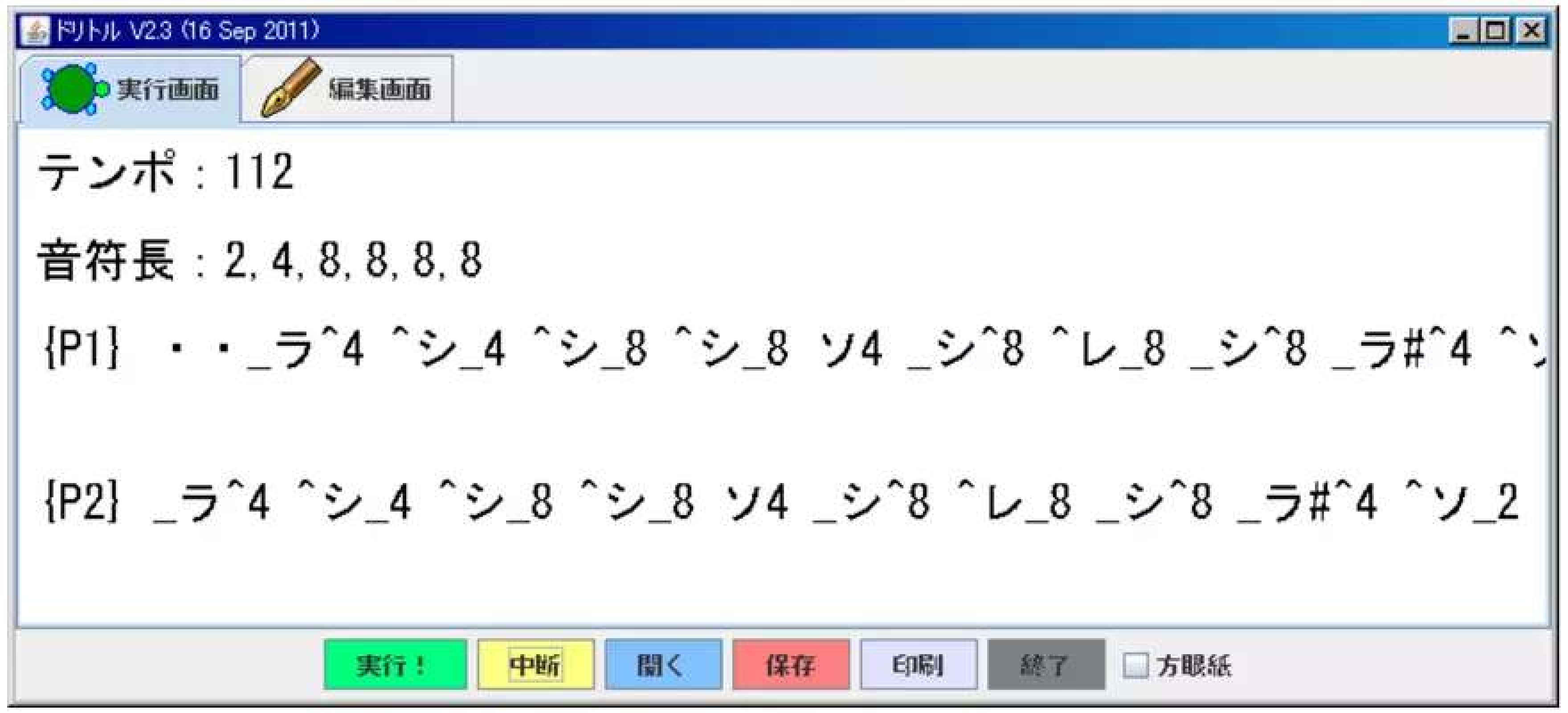
- 使う音符の長さも、長さの組み合わせの中から選べるようにしてみる
- この処理を初期処理に加えてみる

```
初期処理＝[
  システム！ "compose" 使う.
  速さ＝(!("48,64,80,96,112,128") RandomSelect) .
  音符種類=!("2/4/8/8/8/8,2/2/4/4/4/8,2/4/4/4/8/8,2/4/8/16/16/16") RandomSelect.
  使う音符=音符種類! "/" "," 全部置き換える.
].

作曲処理＝[
  小節数＝16. 拍子＝"4/4". 下限＝"↓ソ". 上限＝"ミ↑↑".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  ! (小節数)(拍子)(下限)(上限)(使う音符)(音符配列)(長さ配列) compose2.
  音符数=(音符配列! 要素数?).
  mml_1="". [|i|
    音符＝（音符配列！(i)見る）＋（長さ配列！(i)見る）.
    mml_1＝mml_1＋音符＋" ".
  ]！ (音符数) 繰り返す.
  mml_2="". [|i|
    音符＝（音符配列！(音符数+1-i)見る）＋（長さ配列！(音符数+1-i)見る）.
    mml_2＝mml_2＋音符＋" ".
  ]！ (音符数) 繰り返す.
].

演奏処理＝[
  part_1＝"･･"＋mml_1＋mml_2. part_2＝mml_1＋mml_2＋"･･".
  バンド！ （メロディ！作る (part_1) 追加）追加（メロディ！作る (part_2) 追加）追加 （速
さ/2)テンポ 演奏.
  ラベル！ ("テンポ:"＋速さ) 作る.
  ラベル！ ("音符長:"＋使う音符) 作る 次の行.
  ラベル！ ("{P1} "＋part_1) 作る 次の行.
  ラベル！ ("{P2} "＋part_2) 作る 次の行.
].

！初期処理. ！作曲処理. ！演奏処理.
```



乱数の初期化

- 乱数は初期化をしないとプログラムを実行する度に別の曲になる
- 乱数初期化 (乱数の種) 関数を使うと作品を固定にすることができる
- 変奏番号の値を使って乱数の初期化し作品を固定化してみる
  - 乱数の種＝ 変奏番号*194612

```
初期処理＝[
  システム！ "compose" 使う.
  変奏番号=1.
  乱数初期化 (変奏番号*194612).
  速さ=(!("48,64,80,96,112,128") RandomSelect) .
  音符種類=!("2/4/8/8/8/8,2/2/4/4/4/8,2/4/4/4/8/8,2/4/8/16/16/16") RandomSelect.
  使う音符=音符種類! "/" "," 全部置き換える.
].

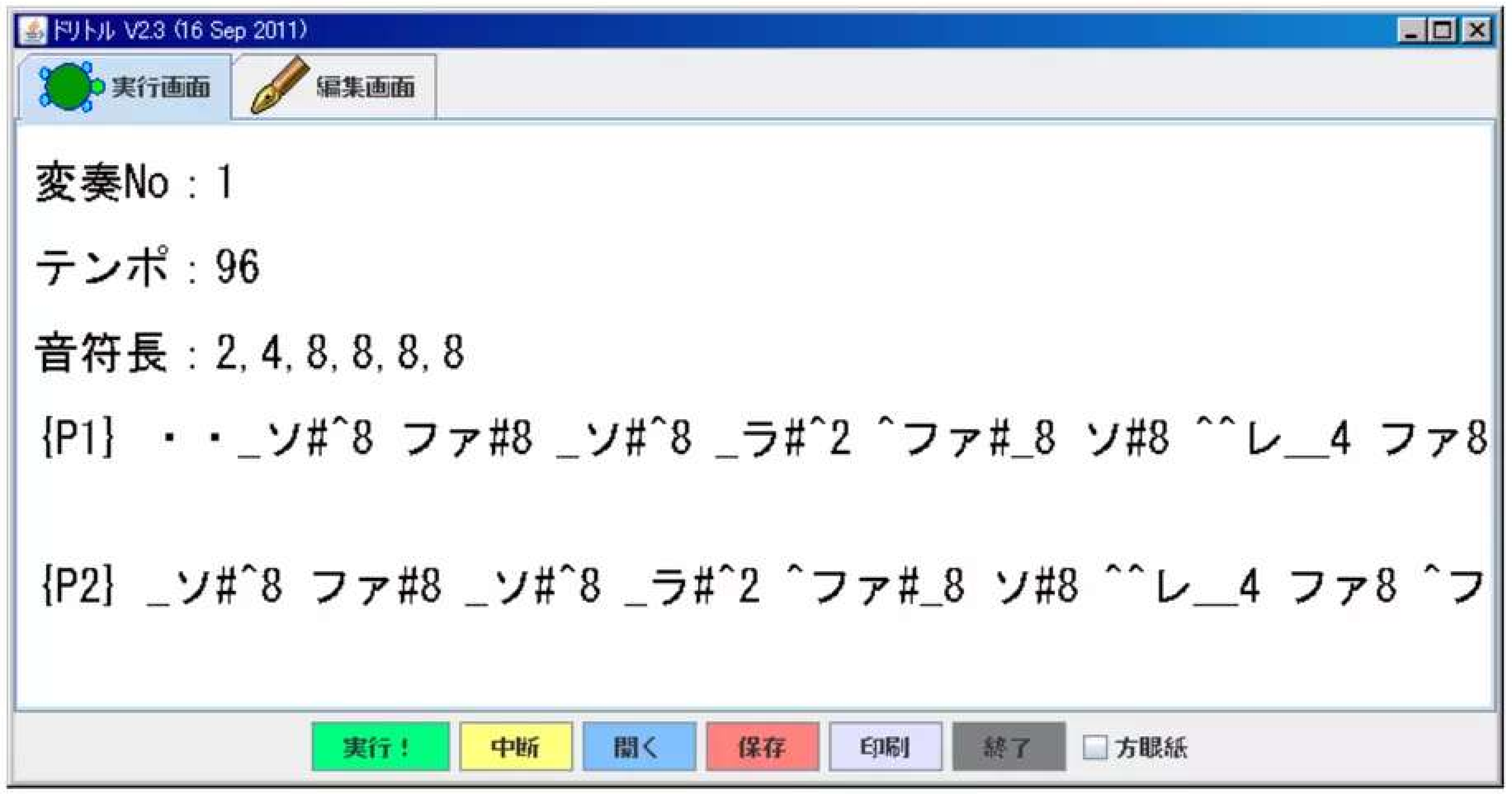
作曲処理＝[

  ・・・・・
  ・・・・・

].

演奏処理＝[
  part_1＝"・・"＋mml_1＋mml_2. part_2＝mml_1＋mml_2＋"・・".
  バンド！ (メロディ！作る (part_1) 追加)追加(メロディ！作る (part_2) 追加)追加 (速さ/2)テンポ 演奏.
  ラベル！ ("変奏No:"＋変奏番号) 作る.
  ラベル！ ("テンポ:"＋速さ) 作る 次の行.
  ラベル！ ("音符長:"＋使う音符) 作る 次の行.
  ラベル！ ("{P1} "＋part_1) 作る 次の行.
  ラベル！ ("{P2} "＋part_2) 作る 次の行.
].

！初期処理. ！作曲処理. ！演奏処理.
```



作品名を考えてみる

- 今まで実験したプログラムを使って音楽作品を作ってみる
- いかにもプログラムで作ったような作品にしましょう
- 回文的なメロディになっているのでタイトルを「Palindrome」にします
- 曲の内容がわかるような副題も付けてみる
- 曲は二つのパートが同じ旋律を追いかける形になっています
- 副題としては「回文」と「カノン」の文字を入れたいところ
- 曲の固定化を1～12の番号でやってみましょう
- 副題は「二声の回文的カノンによる12の変奏」にしてみます
- 12曲あるのでこれをまとめれば作品集（アルバム）になります

Palindrome.dtl の仕様

- 乱数は 194612*変奏番号 で初期化して作品を固定化する
- 曲は二つの声部で構成する
- 各パートは二分の一小節ずれているだけの同じ旋律にする
- 曲は全て32小節の回文的な構造（1～16と17～32で対称）にする
- テンポは、48,64,80,96,112,128 の中からランダムに選ぶ
- 音の長さは、2,4,8,16部音符で構成される音符セットからランダムに選ぶ
- 音の高さは、"↓ソ"～"ミ↑↑"の範囲でランダムに選ぶ
- 12の変奏を連続して演奏するようにする
- どの変奏を演奏しているかを実行画面に表示する

連続して演奏させるには

- 処理手手順を曲数分ループさせる

```
[|v_no|
   ！(v_no) 初期処理. ！作曲処理. ！演奏処理.
]！ (曲数) 繰り返す.
```

- 変奏番号をパラメータにした初期化処理をする

```
初期処理＝[|v_no|
  変奏番号=v_no.
  乱数初期化 (変奏番号*194612).
  速さ=(!("48,64,80,96,112,128") RandomSelect) .
  音符種類=!("2/4/8/8/8/8,2/2/4/4/4/8,2/4/4/4/8/8,2/4/8/16/16/16") RandomSelect.
  使う音符=音符種類! "/" "," 全部置き換える.
  曲の内容＝リスト！作る (画面!幅?)(画面!高さ?) 大きさ ((画面!幅?)*-0.5)((画面!高さ?)*0.5) 位置.
].
```

- １曲演奏が終わるのを待つ処理を入れる

```
  バンド！（メロディ！作る (part_1) 追加 ）追加（メロディ！作る (part_2) 追加）追加 （速さ/2）テンポ 演奏 待つ クリア.
```

Palindrome.dtl を作る

- 『Palindrome』の全曲演奏版の完成です
- 外部音源を使うと上手く演奏できない場合があります

```
// Palindrome.dtl

システム！ "compose" 使う.

タイトル="Palindrome".
曲数=12.
サブタイトル="（二声の回文的カノンによる"+曲数+"の変奏）".

初期処理＝[|v_no|
  変奏番号=v_no.
  乱数初期化 (変奏番号*194612).
  速さ=(!("48,64,80,96,112,128") RandomSelect) .
  音符種類=!("2/4/8/8/8/8,2/2/4/4/4/8,2/4/4/4/8/8,2/4/8/16/16/16") RandomSelect.
  使う音符=音符種類! "/" "," 全部置き換える.
  曲の内容＝リスト！作る (画面!幅?)(画面!高さ?) 大きさ ((画面!幅?)*-0.5)((画面!高
さ?)*0.5) 位置.
].

作曲処理＝[
  小節数＝16. 拍子＝"4/4". 下限＝"↓ソ". 上限＝"ミ↑↑".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  ! (小節数)(拍子)(下限)(上限)(使う音符)(音符配列)(長さ配列) compose2.
  音符数=(音符配列! 要素数?).
  mml_1="". [|i|
    音符＝（音符配列！(i)見る）＋（長さ配列！(i)見る）.
    mml_1＝mml_1＋音符＋" ".
  ]！ (音符数) 繰り返す.
  mml_2="". [|i|
    音符＝（音符配列！(音符数+1-i)見る）＋（長さ配列！(音符数+1-i)見る）.
    mml_2＝mml_2＋音符＋" ".
  ]！ (音符数) 繰り返す.
].
```

```
演奏処理＝[
  part_1＝"・・"＋mml_1＋mml_2. part_2＝mml_1＋mml_2＋"・・".

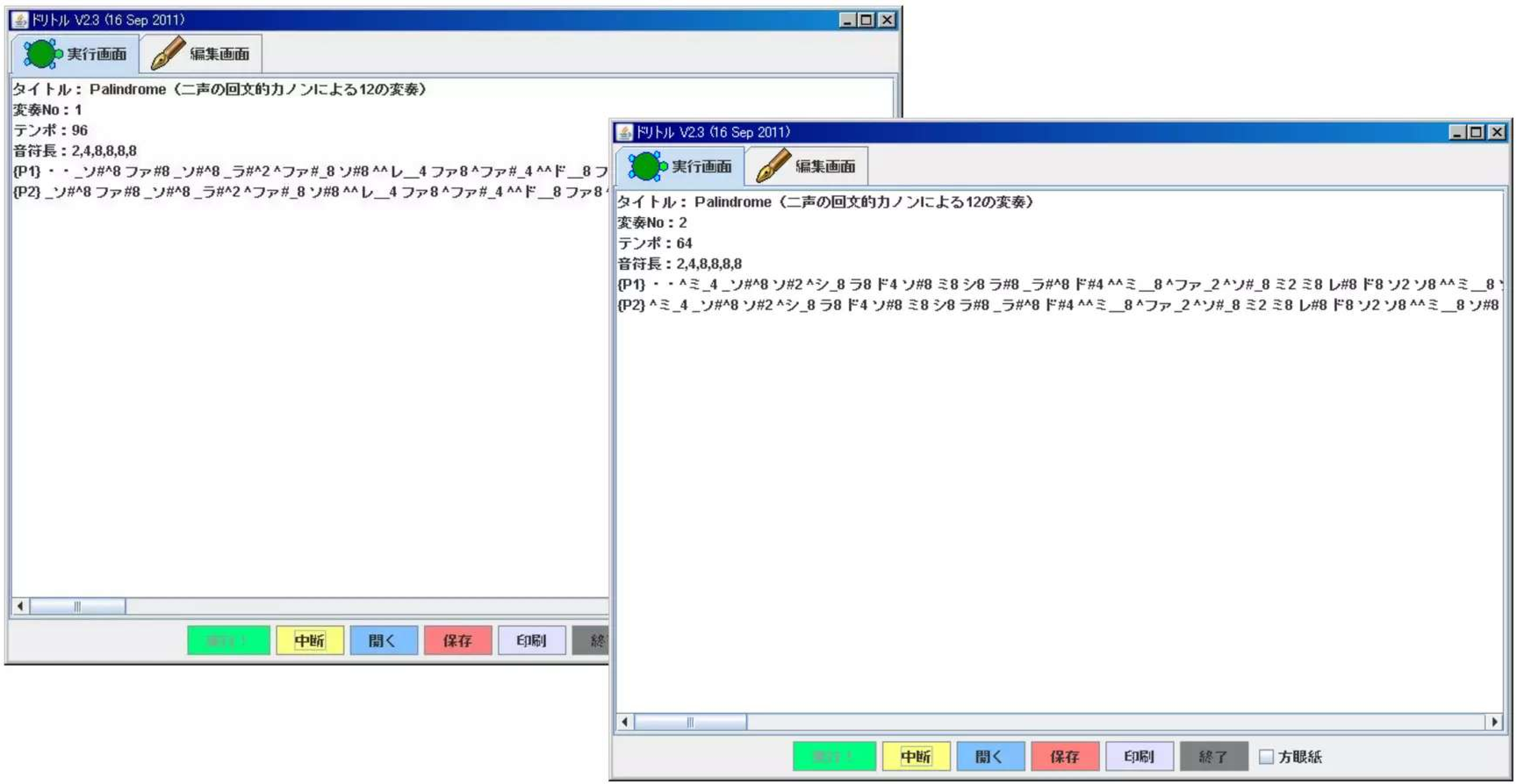
  曲の内容！クリア.
  曲の内容！("タイトル:"＋タイトル＋サブタイトル）書く.
  曲の内容！("変奏No:"＋変奏番号）書く.
  曲の内容！("テンポ:"＋速さ) 書く.
  曲の内容！("音符長:"＋使う音符) 書く.
  曲の内容！("{P1} "＋part_1) 書く.
  曲の内容！("{P2} "＋part_2) 書く.

  バンド！（メロディ！作る (part_1) 追加 ）追加（メロディ！作る (part_2) 追加）追加 （速
さ/2）テンポ 演奏 待つ クリア.
].

[|v_no|
  ！(v_no) 初期処理. ！作曲処理. ！演奏処理.
]！ (曲数) 繰り返す.
```

Palindrome.dtl を実行

- それでは[実行！]ボタンを押して演奏させてみます
- 途中で止める場合は[中断]ボタンを押す



オンライン版ドリトル

- オンライン版ドリトルでもJavaの内蔵音源を使って演奏できる
- ライブラリー部分のソースを一緒に組み込んであります

```
// オンライン版ドリトル用

// 指定範囲の乱数を作る
RandomNumber=[|r_min r_max|
  random(r_max - r_min) + r_min.
].
// カンマ区切りのデータからランダムに選ぶ
RandomSelect=[|slc; ary|
  ary=slc ! "," 分割.  ary! (random(ary! 要素数?)) 読む.
].
// MIDIノート番号からドリトルの音名を作る
NoteName=[|番号; 音階 前 後 octv|
    音階="ド:ド#:レ:レ#:ミ:ファ:ファ#:ソ:ソ#:ラ:ラ#:シ" ! ":" 分割.
    前="_____:____:___:__:_::^:^^:^^^:^^^^:^^^^^]"! ":" 分割.
    後="^^^^^:^^^^:^^^:^^:^::_:__:___:____:_____]"! ":" 分割.
    octv=floor(番号/12)+1.
    (前 ! (octv) 読む) + (音階 ! (番号%12+1) 読む) + (後 ! (octv) 読む).
].
// ドリトルの音名からMIDIノート番号を作る
NoteNumber=[|音名; mns pls onk scl bis k|
  mns=0. pls=0. onk=0.
  scl="c,d,e,f,g,a,b,ド,レ,ミ,ファ,ソ,ラ,シ,#,%,＃,♭" ! "," 分割.
  bis="0,2,4,5,7,9,11,0,2,4,5,7,9,11,1,-1,1,-1" ! "," 分割.
  k=1.[
    [音名! (scl!(k)読む)含む?]! なら [onk=onk+(bis!(k)読む)]実行. k=k+1.
  ] ! (scl! 要素数?) 繰り返す.
  k=1.[
    [(音名 !  (k)(1) 部分)! "↓" 含む?]! なら [mns=mns+1]実行.
    [(音名 !  (k)(1) 部分)! "↑" 含む?]! なら [pls=pls+1]実行.
    k=k+1.
  ] !(音名! 長さ?)繰り返す.
  60 + onk + (pls*12) - (mns*12).
].
compose2＝[|小節数 拍子 下限 上限 使う音符 音符配列 長さ配列; m tt tm to w_base|
  w_base=拍子! "/" 分割.
  m=小節数 * (w_base! (1) 読む) * (4/(w_base! (2) 読む)).
  i=1. tt=0.
  [tt < m] ! の間 [
    note_n=(!(!(!(下限)NoteNumber)(!(上限)NoteNumber)RandomNumber) NoteName).
    tm=(!(使う音符) RandomSelect).
    to=tt. tt=tt+(1/tm)*4.
    [tt > m]! なら [tm=1/(m-to)*4] 実行.
    note_l=tm.
    [(m-to)==0.75]! なら [note_l="8.". tt=m ] 実行.
    [(m-to)==1.50]! なら [note_l="4.". tt=m ] 実行.
    [(m-to)==1.75]! なら [note_l="4..". tt=m ] 実行.
    音符配列! (i)(note_n) 挿入.
    長さ配列! (i)(note_l) 挿入.
    i=i+1.
  ] 実行.
].

タイトル="Palindrome".
曲数=12.
サブタイトル="（二声の回文的カノンによる"+曲数+"の変奏）".

初期処理＝[|v_no|
  変奏番号=v_no.
  乱数初期化 (変奏番号*194612).
  速さ=(!("48,64,80,96,112,128") RandomSelect) .
  音符種類=!("2/4/8/8/8/8,2/2/4/4/4/8,2/4/4/4/8/8,2/4/8/16/16/16") RandomSelect.
  使う音符=音符種類! "/" "," 全部置き換える.
  曲の内容＝リスト ! 作る (画面!幅?)(画面!高さ?) 大きさ ((画面!幅?)*-0.5)((画面!高さ?)*0.5) 位置.
].

作曲処理＝[
  小節数＝16. 拍子＝"4/4". 下限＝"↓ソ". 上限＝"ミ↑↑".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  ! (小節数)(拍子)(下限)(上限)(使う音符)(音符配列)(長さ配列) compose2.
  音符数=(音符配列! 要素数?).
  mml_1="". [|i|
    音符＝（音符配列 ! (i)見る）＋（長さ配列 ! (i)見る）.
    mml_1＝mml_1＋音符＋" ".
  ] !  (音符数) 繰り返す.
  mml_2="". [|i|
    音符＝（音符配列 ! (音符数+1-i)見る）＋（長さ配列 ! (音符数+1-i)見る）.
    mml_2＝mml_2＋音符＋" ".
  ] !  (音符数) 繰り返す.
].

演奏処理＝[
  part_1＝"・・"＋mml_1＋mml_2. part_2＝mml_1＋mml_2＋"・・".

  曲の内容 ! クリア.
  曲の内容 ! ("タイトル："＋タイトル＋サブタイトル）書く.
  曲の内容 ! ("変奏No："＋変奏番号）書く.
  曲の内容 ! ("テンポ："＋速さ) 書く.
  曲の内容 ! ("音符長："＋使う音符) 書く.
  曲の内容 ! ("{P1} "＋part_1) 書く.
  曲の内容 ! ("{P2} "＋part_2) 書く.

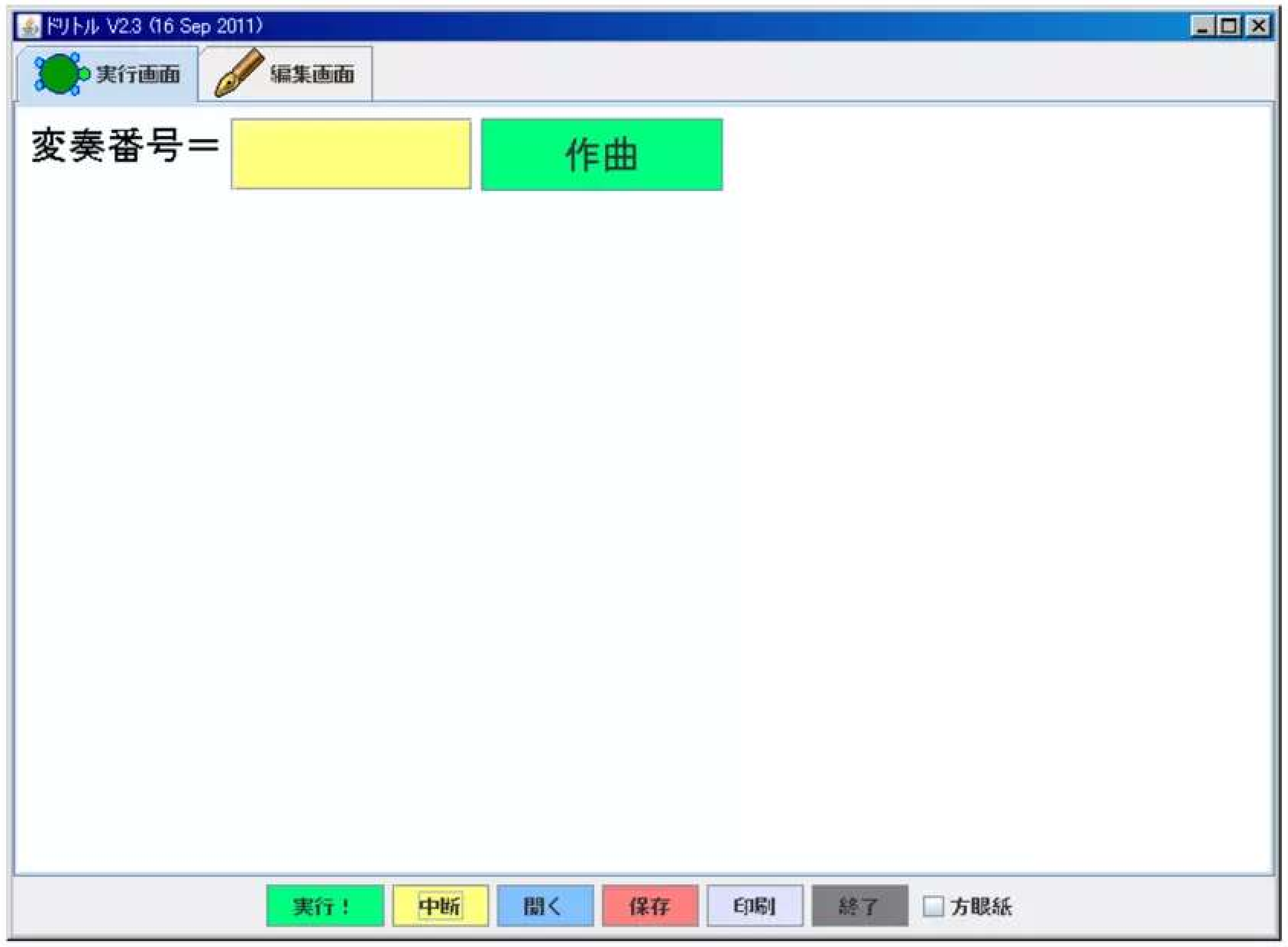
  バンド ! （メロディ ! 作る (part_1) 追加 ）追加（メロディ ! 作る (part_2) 追加）追加 （速さ/2）テンポ 演奏 待つ クリア.
].

[|v_no|
   ! (v_no) 初期処理.  ! 作曲処理.  ! 演奏処理.
] !  (曲数) 繰り返す.
```

作曲ボタンを作る

- ドリトルはイベントドリブンな処理を簡単に記述できる
- ・[作曲]ボタンを押すとプログラムを実行できるようにしてみる
- 変奏番号はフィールドを作って取得
- ・メイン処理を書き換えるだけで完成！

```
ラベル！ "変奏番号＝" 作る.
変奏番号＝フィールド！作る. 作曲実行＝ボタン！ "作曲" 作る.
作曲実行:動作＝[
！（変奏番号！読む）初期処理. ！作曲処理. ！演奏処理.
].
```



演奏楽器の指定

- 演奏楽器も指定もできるようにしてみましょう
- パート毎に楽器を指定できるようにしてみる

```
// Palindrome.dtl（作曲ボタン付き）

システム！ "compose" 使う.

タイトル="Palindrome".
曲数=12.

初期処理＝[|v_no|
  変奏番号=v_no.
  乱数初期化 (変奏番号*194612).
  速さ=(!("48,64,80,96,112,128") RandomSelect) .
  音符種類=!("2/4/8/8/8/8,2/2/4/4/4/8,2/4/4/4/8/8,2/4/8/16/16/16") RandomSelect.
  使う音符=音符種類! "/" "," 全部置き換える.
  曲の内容＝リスト！作る (画面!幅?)(画面!高さ?) 大きさ ((画面!幅?)*-0.5)((画面!高さ?)*0.5) 位置.
].

作曲処理＝[
  小節数＝16. 拍子＝"4/4". 下限＝"↓ソ". 上限＝"ミ↑↑".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  ! (小節数)(拍子)(下限)(上限)(使う音符)(音符配列)(長さ配列) compose2.
  音符数=(音符配列! 要素数?).
  mml_1="". [|i|
    音符＝（音符配列！(i)見る）＋（長さ配列！(i)見る）.
    mml_1＝mml_1＋音符＋" ".
  ]！ (音符数) 繰り返す.
  mml_2="". [|i|
    音符＝（音符配列！(音符数+1-i)見る）＋（長さ配列！(音符数+1-i)見る）.
    mml_2＝mml_2＋音符＋" ".
  ]！ (音符数) 繰り返す.
].
```

```
演奏処理＝[|楽器1 楽器2|
  part_1＝"・・"＋mml_1＋mml_2. part_2＝mml_1＋mml_2＋"・・".

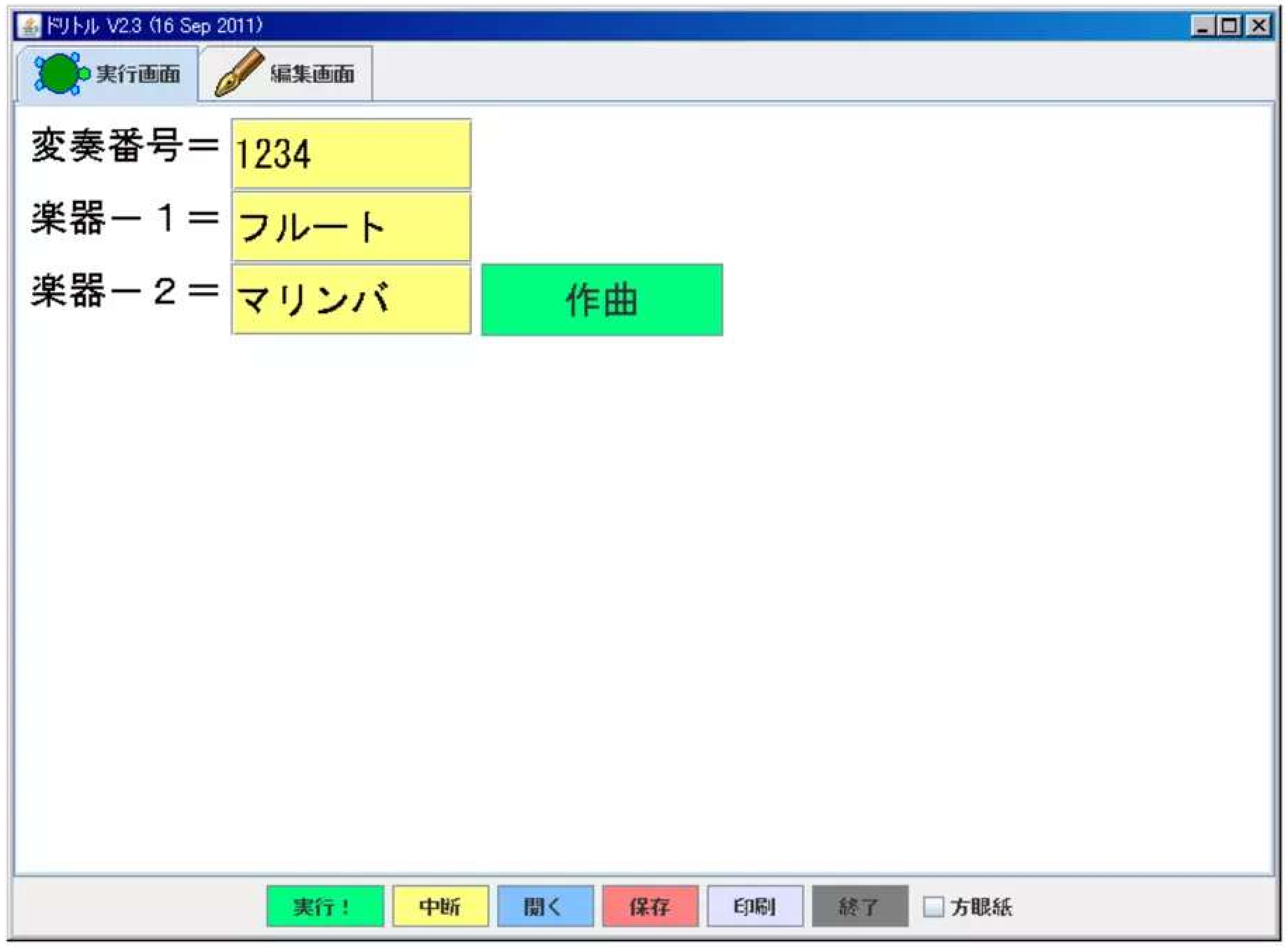
  曲の内容！クリア.
  曲の内容！("タイトル:"＋タイトル＋サブタイトル）書く.
  曲の内容！("変奏No:"＋変奏番号）書く.
  曲の内容！("テンポ:"＋速さ) 書く.
  曲の内容！("音符長:"＋使う音符) 書く.
  曲の内容！("演奏楽器:"＋楽器1＋" / "＋楽器2) 書く.
  曲の内容！("{P1} "＋part_1) 書く.
  曲の内容！("{P2} "＋part_2) 書く.

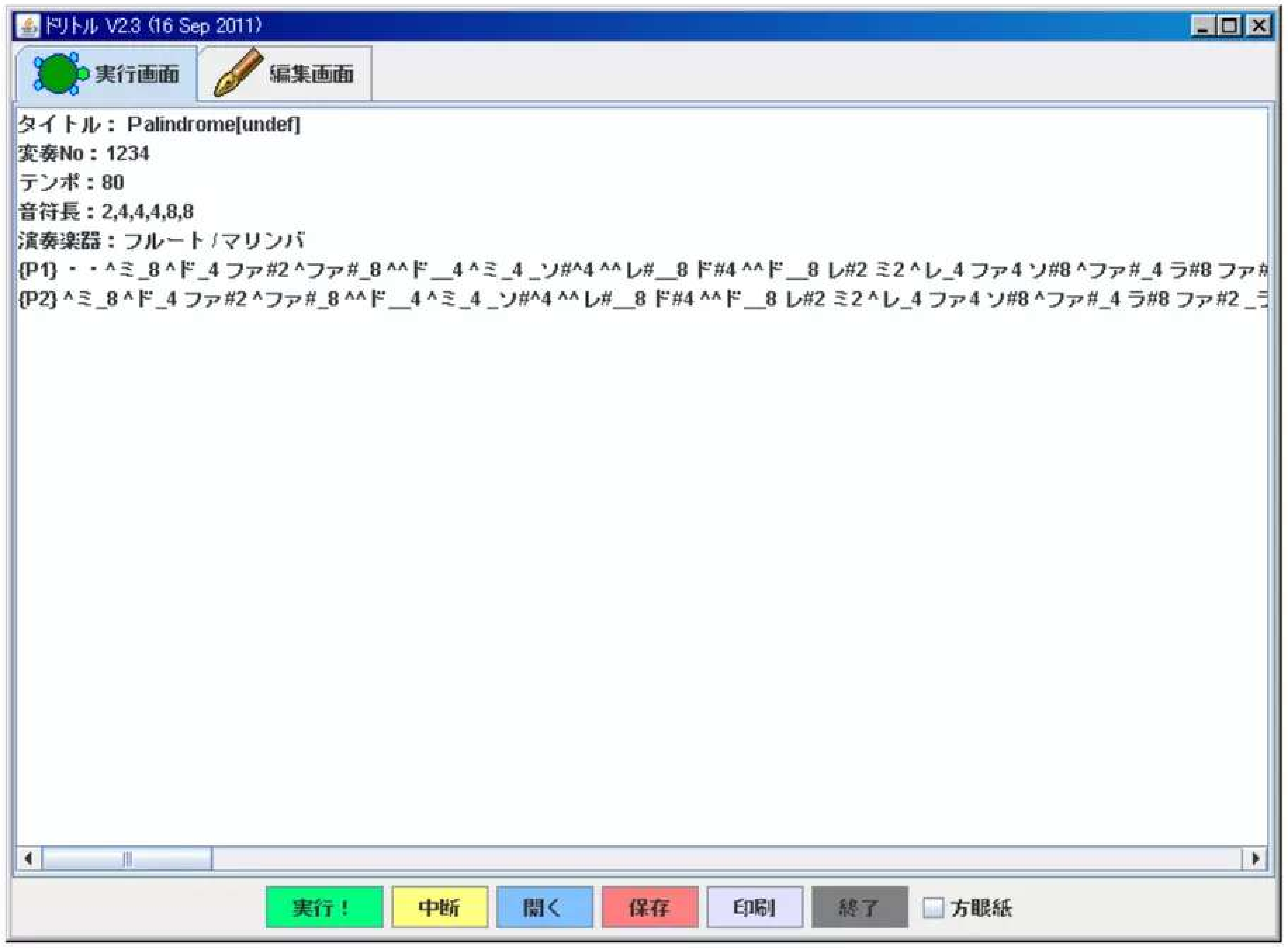
  inst_1＝楽器！（楽器1）作る. inst_2＝楽器！（楽器2）作る.
  バンド！（メロディ！作る (inst_1) 設定 (part_1) 追加）追加（メロディ！作る (inst_2) 設定 (part_2) 追加）追加 （速さ/2）テンポ 演奏.
].

ラベル！ "変奏番号＝" 作る. 変奏番号＝フィールド！作る.
ラベル！ "楽器－1＝" 作る 次の行. 楽器1＝フィールド！作る.
ラベル！ "楽器－2＝" 作る 次の行. 楽器2＝フィールド！作る.
作曲実行＝ボタン！ "作曲" 作る.
作曲実行:動作＝[
 ！（変奏番号！読む）初期処理. ！作曲処理. ！（楽器1！読む）（楽器2！読む）演奏処理.
].
```

楽器を指定して演奏

- 思い付いた楽器名を適当に入れて演奏させてみましょう
- 楽器名がドリトルで定義されていないとピアノで演奏します
- 意外な楽器の組み合わせが面白かったりします





Ⅲ 作曲した音楽を楽譜にする

楽譜とMIDIができると

- ドリトルはMIDI音源を使って演奏できるのだが
- ドリトルだけで楽譜やMIDIデータは作れない
- LilyPond というシステムを使いましょう
  - LilyPond はフリーソース
  - ドリトルと同じようにMACやLinuxでも動く
- 楽譜ができれば
  - ドリトルで作曲した作品を出版することができる
  - 生の楽器で演奏してもらうことができる
- MIDIデータができれば
  - MIDIデータを加工して表現力のある曲に仕上げられる
  - DAWを使って録音しMP3やCDなどにして聴いてもらうことができる

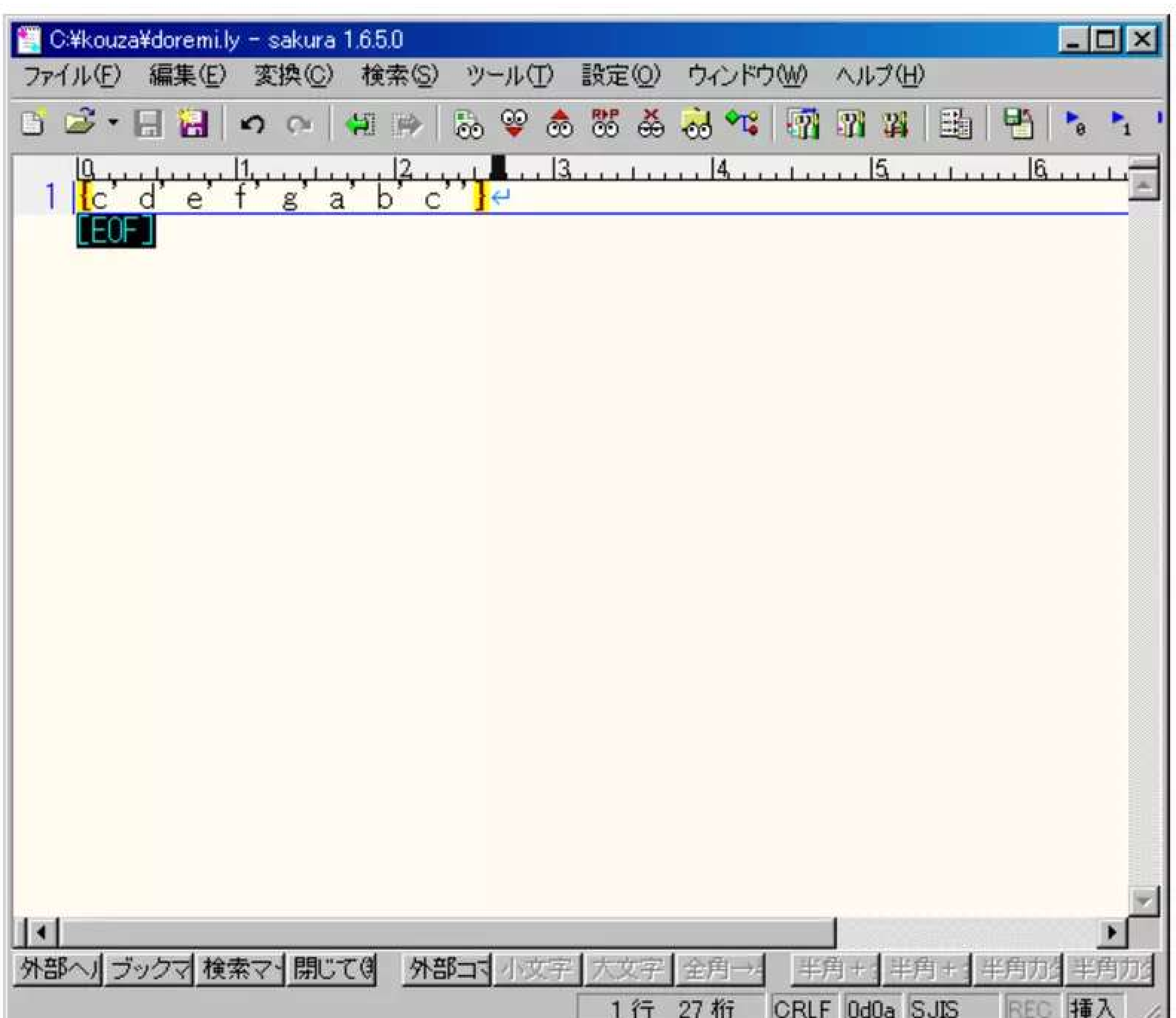
LilyPondのインストール

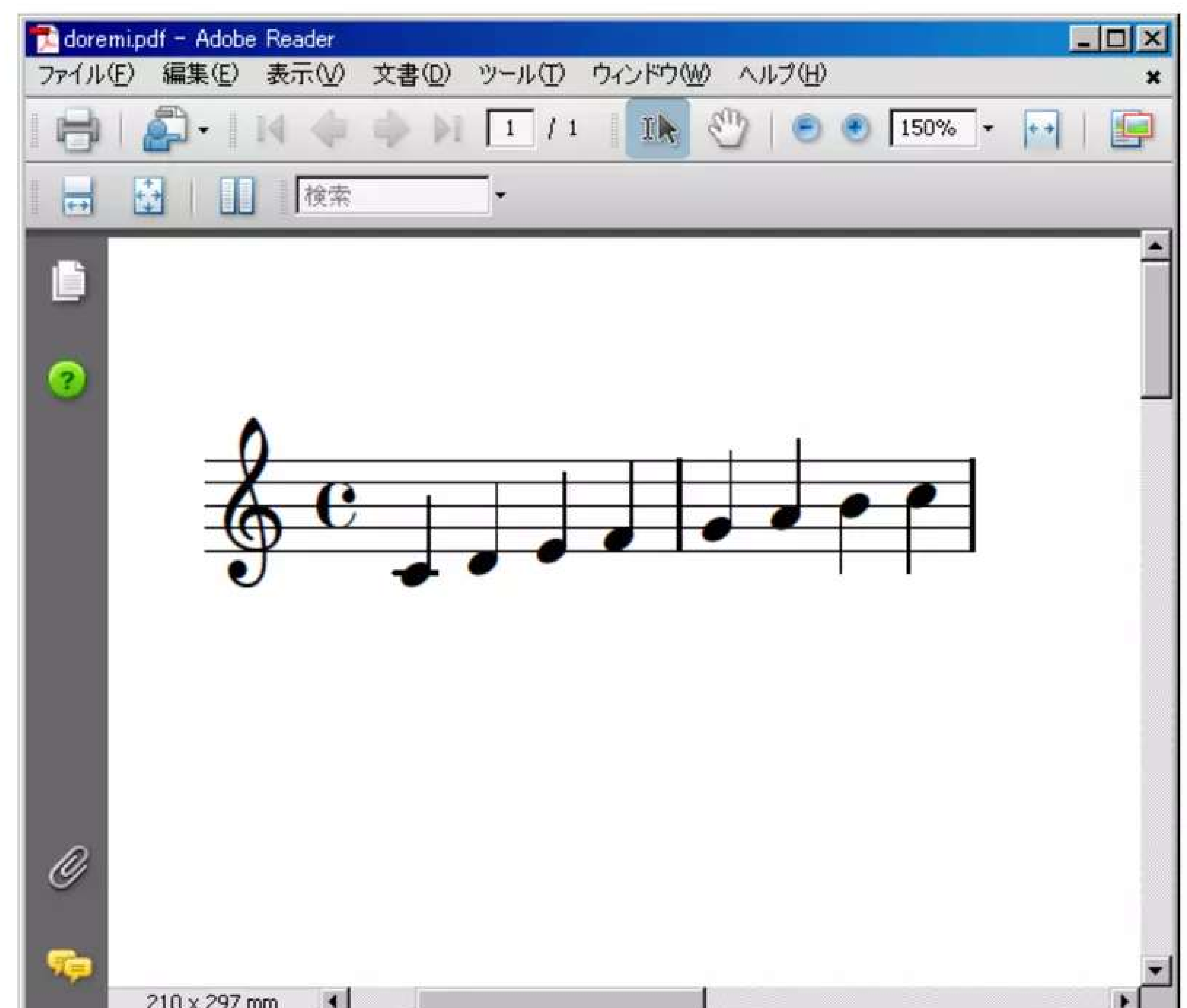
- LilyPondのサイトからインストールデータを取得
  - http://lilypond.org/index.ja.html



LilyPondを実行してみる

- LilyPondのMMLはドリトルとほとんど同じ
  - ドリトル　『ド レ ミ ファ ソ ラ シ ↑ド』
  - lilypond　{c' d' e' f' g' a' b' c''}
- LilyPondソース（doremi.ly）を作成＆保存
- doremi.lyを実行（楽譜のPDFができる！）





LilyPondコードの構造

- LilyPondのソースコードはこんな形式
- 複合音楽表記 の部分にLilyPond用のMMLを記述

```
\version "2.12.3"

\header { }

\score {
  複合音楽表記
  \layout { }
  \midi { }
}.
```

- ･LilyPond用のMMLを作るメソッドを作成

```
// MIDIノート番号からLilyPondの音名を作る
NoteNameL=[|番号; 音階 octv|
  音階="c:cis:d:dis:e:f:fis:g:gis:a:ais:b" ! ":" 分割.
  octv=",,,,:,,,:,,:,::':'':''':'''':''''']"! ":" 分割.
  (音階 ! (番号%12+1) 読む) + (octv ! (floor(番号/12)+1) 読む).
].
```

コードの型ファイル

- 2パートの曲を記述するための一般的な構造を定義
- この中に曲により可変になる部分を変数にしてみる
- 変数は %変数名 のように書いておく

```
\version "2.12.3"
\paper {print-page-number = ##f}
\header {title="%title" subtitle="%subtitle" subsubtitle="%subsubtitle" composer="A.Yamazawa"  copyright=" "  tagline=" " }
\score {
<<
\new Staff
<<
\set Staff.instrumentName = #"Part 1"
\set Staff.midiInstrument = #"acoustic grand"
\clef "treble"
\time %time
\tempo 4 = %tempo
\new Voice = ""
\with {\remove "Note_heads_engraver" \consists "Completion_heads_engraver" }
{%part_1 \bar"|."}
>>
\new Staff
<<
\set Staff.instrumentName = #"part 2"
\set Staff.midiInstrument = #"acoustic grand"
\clef "treble"
\new Voice = ""
\with {\remove "Note_heads_engraver" \consists "Completion_heads_engraver" }
{%part_2 \bar"|."}
>>
>>
\layout {}
\midi {}
}
```

ドリトルのテキスト処理

- ドリトルのテキスト処理はかなり高度
- LilyPondのソースコードを作るには
  - 定義ファイル（frame.ly）を読む
  - 定義ファイルの変数の部分を置き換える
  - 置換したファイル（palindrome.ly）を書く

```
lly_f＝テキストファイル！ "frame.ly" 作る.
f_wrk＝(lly_f！ 読む)! "｜" 結合.

f_tmp＝f_wrk! "%title" (タイトル)        置き換える. f_wrk=f_tmp.
f_tmp＝f_wrk! "%subtitle" ("No."＋変奏番号) 置き換える. f_wrk=f_tmp.
f_tmp＝f_wrk! "%subsubtitle" (" ")       置き換える. f_wrk=f_tmp.
f_tmp＝f_wrk! "%time" (拍子)             置き換える. f_wrk=f_tmp.
f_tmp＝f_wrk! "%tempo" (速さ)            置き換える. f_wrk=f_tmp.
f_tmp＝f_wrk! "%part_1" (part_1)         置き換える. f_wrk=f_tmp.
f_tmp＝f_wrk! "%part_2" (part_2)         置き換える. f_wrk=f_tmp.
lly_t＝テキストファイル！ "palindrome.ly") 作る.

lly_t! (f_tmp! "｜" 分割) 全部書く.
```

ソースファイルを生成する

- 作るMMLがドリトル用かLilyPomd用かをパラメータで指定
- 楽譜作成用のボタンを作りこれが押されたら実行

```
// Palindrome.dtl（作曲・楽譜作成＆演奏）

システム！ "compose" 使う.

タイトル="Palindrome".
曲数=12.
サブタイトル="（二声の回文的カノンによる"+曲数+"の変奏）".

初期処理＝[|v_no|
  変奏番号=v_no.
  乱数初期化 (変奏番号*194612).
  速さ=(!("48,64,80,96,112,128") RandomSelect) .
  音符種類=!("2/4/8/8/8/8,2/2/4/4/4/8,2/4/4/4/8/8,2/4/8/16/16/16") RandomSelect.
  使う音符=音符種類! "/" "," 全部置き換える.
  曲の内容＝リスト！作る (画面!幅?)(画面!高さ?) 大きさ ((画面!幅?)*-0.5)((画面!高
さ?)*0.5) 位置.
].

作曲処理＝[|PS|
  小節数＝16. 拍子＝"4/4". 下限＝"↓ソ". 上限＝"ミ↑↑".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  ! (小節数)(拍子)(下限)(上限)(使う音符)(音符配列)(長さ配列)(PS) compose2.
  音符数=(音符配列! 要素数?).
  mml_1="". [|i|
    音符＝（音符配列！(i)見る）＋（長さ配列！(i)見る）.
    mml_1＝mml_1＋音符＋" ".
  ]！ (音符数) 繰り返す.
  mml_2="". [|i|
    音符＝（音符配列！(音符数+1-i)見る）＋（長さ配列！(音符数+1-i)見る）.
    mml_2＝mml_2＋音符＋" ".
  ]！ (音符数) 繰り返す.
].
```

```
演奏処理＝[|PS|
  [PS=="P"]! なら [
    part_1＝"・・"＋mml_1＋mml_2. part_2＝mml_1＋mml_2＋"・・".
      バンド！（メロディ！作る (part_1) 追加 ）追加（メロディ！作る (part_2) 追加）追加
（速さ/2）テンポ 演奏 待つ クリア.
  ]実行.

  [PS=="S"]! なら [
    part_1＝"r2 "＋mml_1＋mml_2. part_2＝mml_1＋mml_2＋"r2".

    lly_f＝テキストファイル！ "frame.ly" 作る.
    f_wrk＝(lly_f！ 読む)! "｜" 結合.
    f_tmp＝f_wrk! "%title" (タイトル)        置き換える. f_wrk=f_tmp.
    f_tmp＝f_wrk! "%subtitle" ("No."＋変奏番号) 置き換える. f_wrk=f_tmp.
    f_tmp＝f_wrk! "%subsubtitle" (" ")       置き換える. f_wrk=f_tmp.
    f_tmp＝f_wrk! "%time" (拍子)             置き換える. f_wrk=f_tmp.
    f_tmp＝f_wrk! "%tempo" (速さ)            置き換える. f_wrk=f_tmp.
    f_tmp＝f_wrk! "%part_1" (part_1)         置き換える. f_wrk=f_tmp.
    f_tmp＝f_wrk! "%part_2" (part_2)         置き換える. f_wrk=f_tmp.
    lly_t＝テキストファイル！ "palindrome.ly" 作る.
    lly_t! (f_tmp! "｜" 分割) 全部書く.
  ]実行.
].

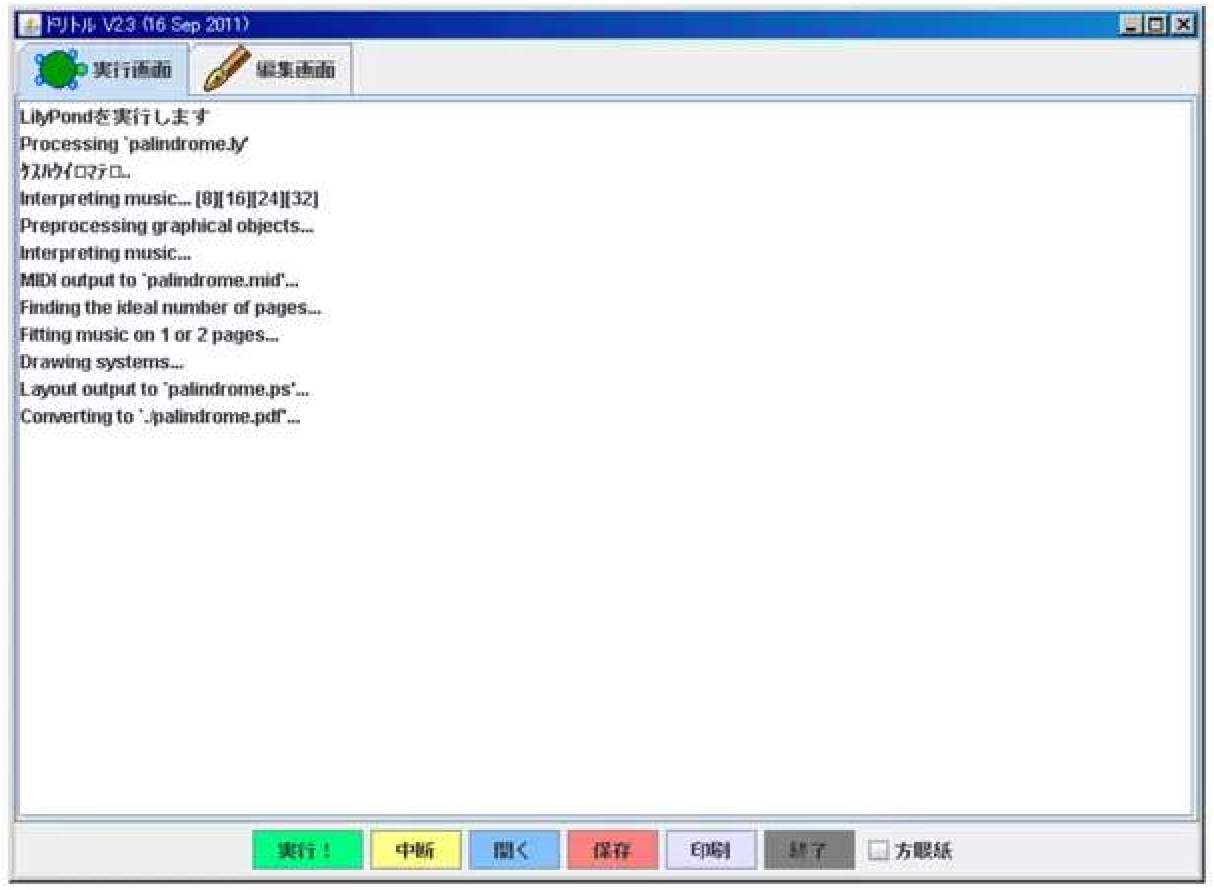
ラベル！ "変奏番号＝" 作る.
変奏番号＝フィールド！作る.
作曲実行＝ボタン！ "作曲" 作る. 楽譜作成＝ボタン！ "楽譜" 作る.
作曲実行:動作＝[
！（変奏番号！読む）初期処理. ！("P") 作曲処理. ！("P") 演奏処理.
].
楽譜作成:動作＝[
！（変奏番号！読む）初期処理. ！("S") 作曲処理. ！("S") 演奏処理.
].
```

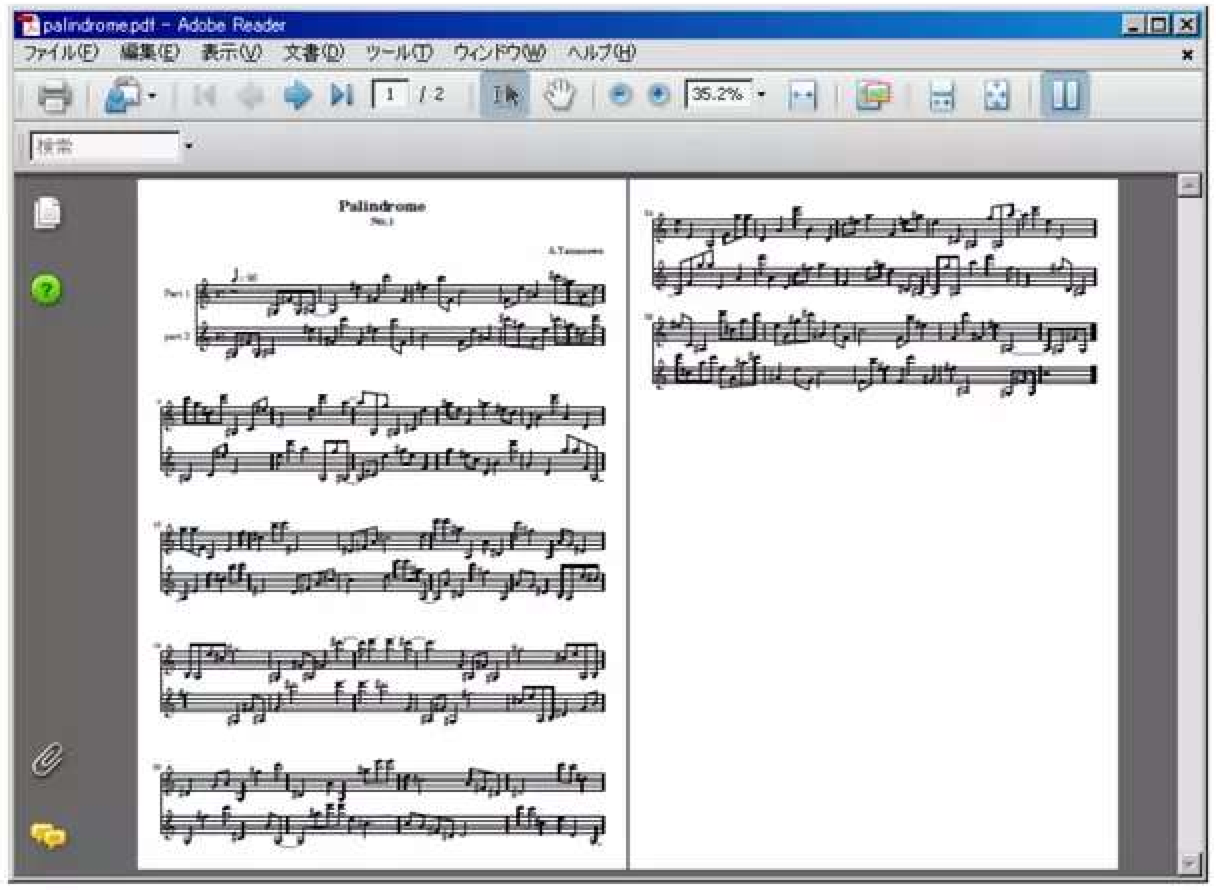
楽譜を表示させる

- LilyPondのコンパイルをドリトルから実行してみる
- システム！ オブジェクトを使います
- コンパイル結果を取得してドリトルの画面に表示させてみる
- 出来上がった楽譜（PDF）も自動的に表示させるようにしてみましょう

```
曲の内容！クリア.
曲の内容！("LilyPondを実行します")書く.
結果＝システム！ ("lilypond palindrome.ly") 実行.
msg=(結果:stderr)! "\n" 分割.
[|i| 曲の内容！(msg! (i) 読む)書く]！(msg! 要素数?) 繰り返す.
システム！ ("cmd /c  palindrome.pdf") 実行.
```



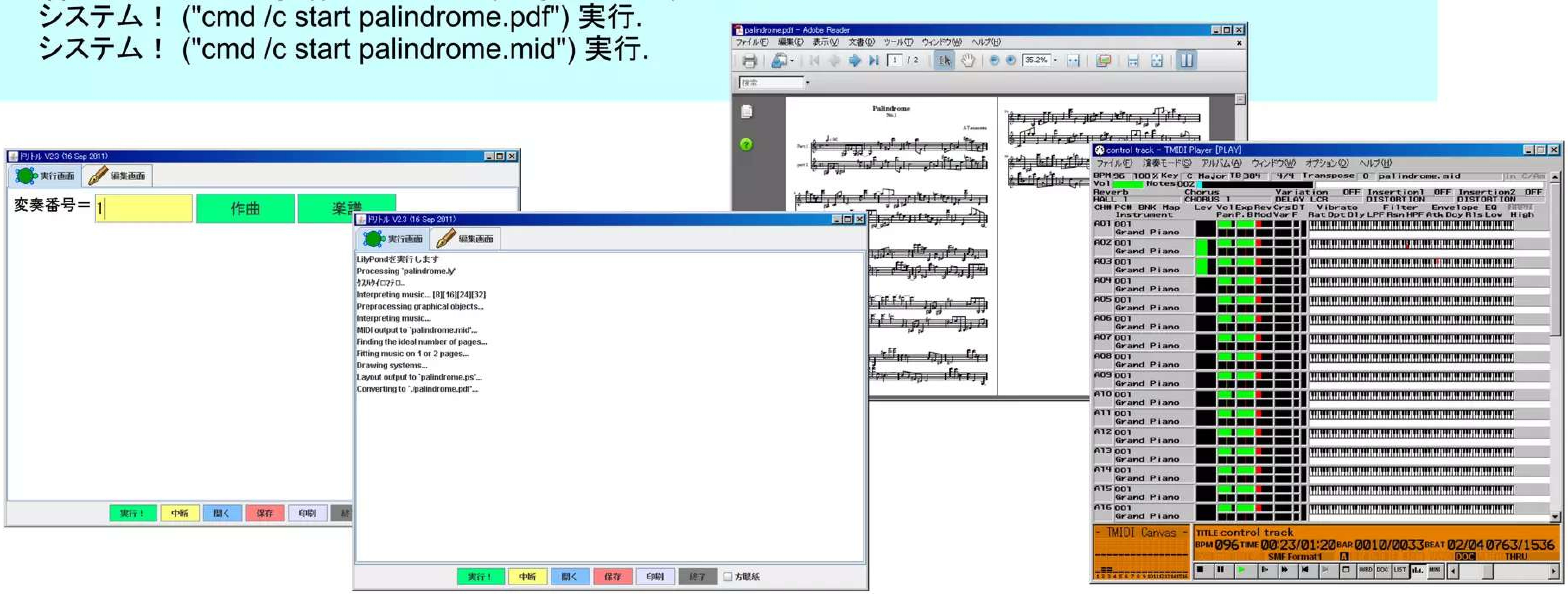




MIDIを演奏させる

- LilyPondが生成したMIDIデータをも演奏させてみる
- 拡張子が .mid に関連づけされているアプリがあれば自動的に演奏する
- 楽譜を見ながら演奏＆チェック

```
曲の内容！クリア.
曲の内容！("LilyPondを実行します")書く.
結果＝システム！　("lilypond "+変奏番号+".ly")  実行.
msg=(結果:stderr)! "\n" 分割.
[|i| 曲の内容！(msg! (i) 読む)書く]！(msg! 要素数?) 繰り返す.
システム！ ("cmd /c start palindrome.pdf") 実行.
システム！ ("cmd /c start palindrome.mid") 実行.
```



名前を変えて書き出す

- 全曲の楽譜を作る実行ボタンを追加
- 全曲ボタンを押すと1～変奏番号までの楽譜を作成

```
楽譜全曲＝ボタン！ "全曲" 作る.
楽譜全曲：動作＝[
  曲数＝変奏番号！読む.
  [|i|！（i）初期処理. ！("S") 作曲処理. ！("S") 演奏処理.]!（曲数）繰り返す.
].
```

- LilyPondのソースコードのファイル名は 番号.ly のようにする

```
lly_t＝テキストファイル！ (変奏番号+".ly") 作る.

システム！ ("lilypond "+変奏番号+".ly") 実行.
```

全曲楽譜生成プログラム

- Palindrome の全曲楽譜生成機能付きのソースコード

```
// Palindrome.dtl（全曲楽譜生成機能付き）

システム！ "compose" 使う.

タイトル="Palindrome".
曲数=12.
サブタイトル="（二声の回文的カノンによる"+曲数+"の変奏）".

初期処理＝[|v_no|
  変奏番号=v_no.
  乱数初期化 (変奏番号*194612).
  速さ=(!("48,64,80,96,112,128") RandomSelect) .
  音符種類=!("2/4/8/8/8/8,2/2/4/4/4/8,2/4/4/4/8/8,2/4/8/16/16/16") RandomSelect.
  使う音符=音符種類! "/" "," 全部置き換える.
  曲の内容＝リスト！作る (画面!幅?)(画面!高さ?) 大きさ ((画面!幅?)*-0.5)((画面!高さ?)*0.5) 位置.
].

作曲処理＝[|PS|
  小節数＝16. 拍子＝"4/4". 下限＝"↓ソ". 上限＝"ミ↑↑".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  !(小節数)(拍子)(下限)(上限)(使う音符)(音符配列)(長さ配列)(PS) compose2.
  音符数=(音符配列! 要素数?).
  mml_1="". [|i|
    音符＝（音符配列！(i)見る）＋（長さ配列！(i)見る）.
    mml_1＝mml_1＋音符＋" ".
  ]！ (音符数) 繰り返す.
  mml_2="". [|i|
    音符＝（音符配列！(音符数+1-i)見る）＋（長さ配列！(音符数+1-i)見る）.
    mml_2＝mml_2＋音符＋" ".
  ]！ (音符数) 繰り返す.
].

演奏処理＝[|PS|
  [PS=="P"]! なら [
    part_1＝"・・"＋mml_1＋mml_2. part_2＝mml_1＋mml_2＋"・・".

    曲の内容！クリア.
    曲の内容！("タイトル："＋タイトル＋サブタイトル）書く.
    曲の内容！("変奏No："＋変奏番号）書く.
    曲の内容！("テンポ："＋速さ) 書く.
    曲の内容！("音符長："＋使う音符) 書く.
    曲の内容！("{P1} "＋part_1) 書く.
    曲の内容！("{P2} "＋part_2) 書く.
```

```
    バンド！（メロディ！作る (part_1) 追加 ）追加（メロディ！作る (part_2) 追加）追加 （速さ/2）テンポ
演奏 待つ クリア.
  ]実行.

  [PS=="S"]! なら [
    part_1＝"r2 "＋mml_1＋mml_2. part_2＝mml_1＋mml_2＋"r2".

    lly_f＝テキストファイル！ "frame.ly" 作る.
    f_wrk＝(lly_f！ 読む)! "｜" 結合.
    f_tmp＝f_wrk! "%title" (タイトル)        置き換える. f_wrk=f_tmp.
    f_tmp＝f_wrk! "%subtitle" ("No."＋変奏番号) 置き換える. f_wrk=f_tmp.
    f_tmp＝f_wrk! "%subsubtitle" (" ")       置き換える. f_wrk=f_tmp.
    f_tmp＝f_wrk! "%time" (拍子)             置き換える. f_wrk=f_tmp.
    f_tmp＝f_wrk! "%tempo" (速さ)            置き換える. f_wrk=f_tmp.
    f_tmp＝f_wrk! "%part_1" (part_1)         置き換える. f_wrk=f_tmp.
    f_tmp＝f_wrk! "%part_2" (part_2)         置き換える. f_wrk=f_tmp.
    lly_t＝テキストファイル！ (変奏番号+".ly") 作る.
    lly_t! (f_tmp! "｜" 分割) 全部書く.

    曲の内容！クリア.
    曲の内容！("LilyPondを実行します"）書く.
    結果＝システム！ ("lilypond "+変奏番号+".ly")  実行.
    msg=(結果:stderr)! "\n" 分割.
    [|i| 曲の内容！（msg! (i) 読む）書く]！(msg! 要素数?) 繰り返す.
    システム！ ("cmd /c start "+変奏番号+".pdf") 実行.
    システム！ ("cmd /c start "+変奏番号+".mid") 実行
  ]実行.
].

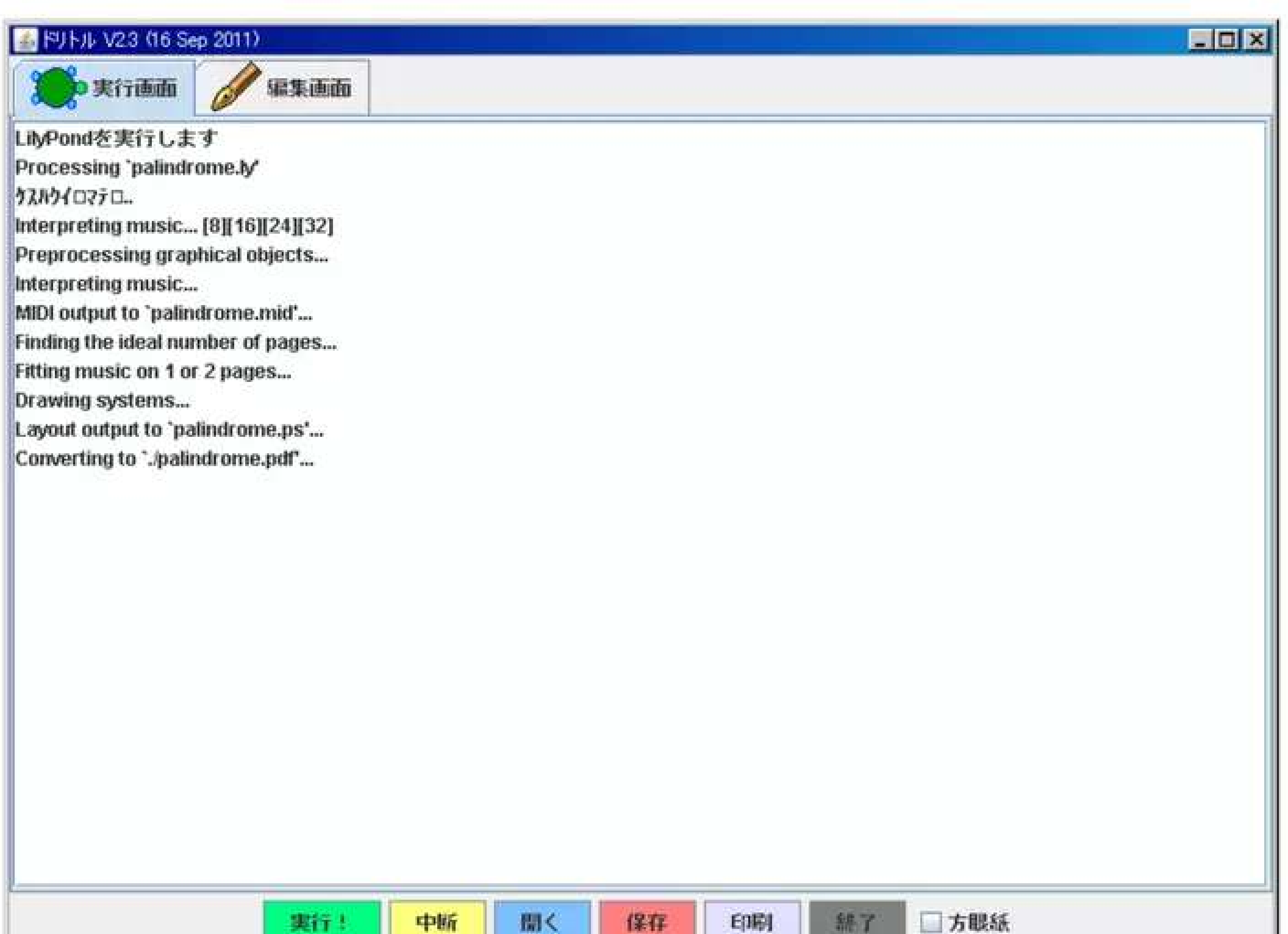
ラベル！ "変奏番号＝" 作る. 変奏番号＝フィールド！作る. 作曲実行＝ボタン！ "作曲" 作る.
楽譜作成＝ボタン！ "楽譜" 作る. 楽譜全曲＝ボタン！ "全曲" 作る.

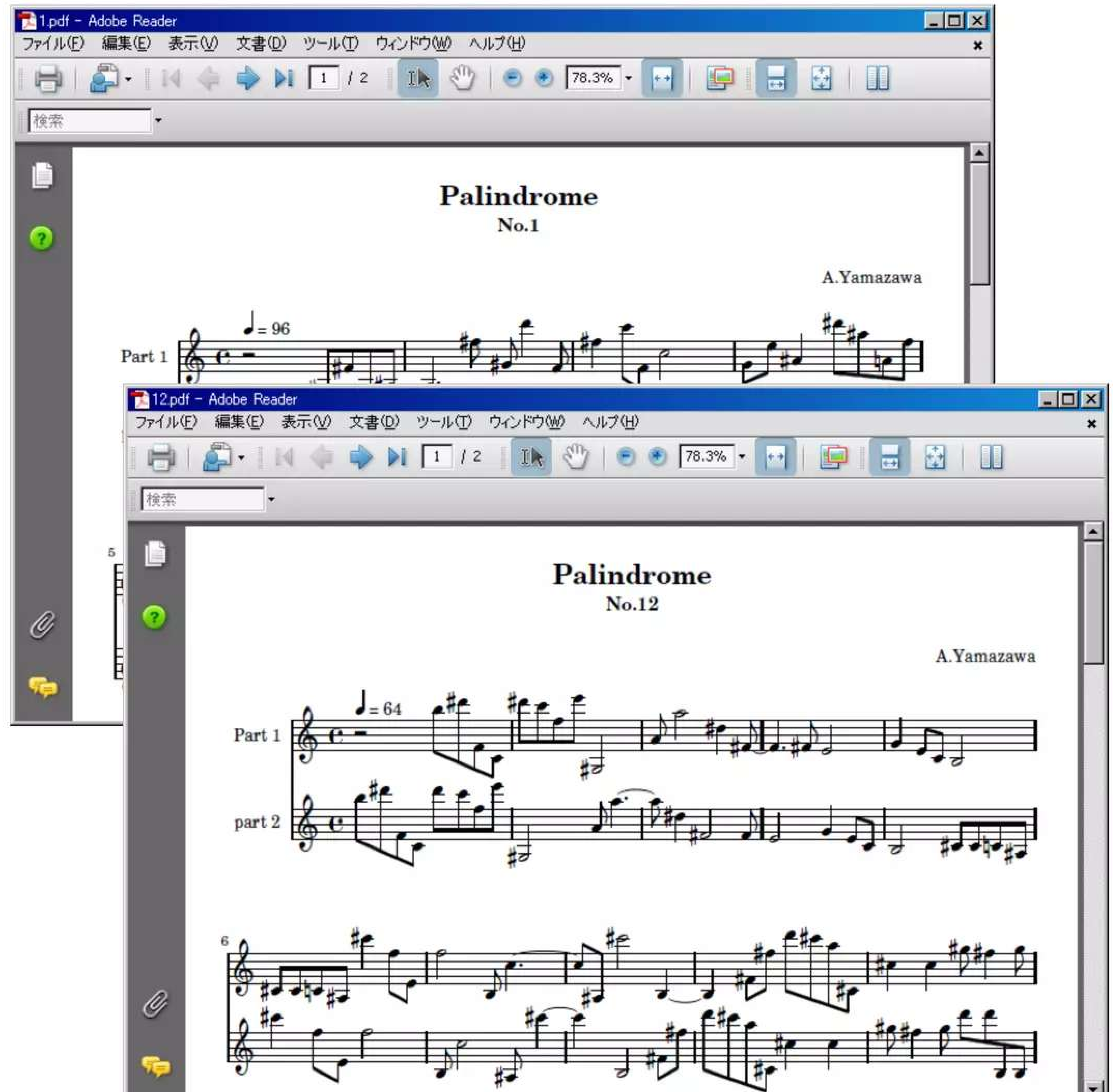
作曲実行：動作＝[
 ！（変奏番号！読む）初期処理. ！("P") 作曲処理. ！("P") 演奏処理.
].
楽譜作成：動作＝[
 ！（変奏番号！読む）初期処理. ！("S") 作曲処理. ！("S") 演奏処理.
].
楽譜全曲：動作＝[
  曲数＝変奏番号！読む.
  [|i|！(i) 初期処理. ！("S") 作曲処理. ！("S") 演奏処理.]!（曲数）繰り返す.
].
```

全曲の楽譜&MIDI作成

- No,1～12までも楽譜（pdf）とMIDIデータを作ってみる







スクリプト言語として使う

- どうすればできるかを考えてみる
  - パラメータをファイルから読む
  - 自動的にドリトルを終了させる

```
prf=textfile! “prm.txt” create.
変奏番号=((prf! get)! 1 get).

！（変奏番号）初期処理. ！("S") 作曲処理. ！楽譜作成.

システム! (はい) 終了する.
```

バッチ用Palindrome.dtl

- バッチ用のプログラム（palindromeB.dtl）を作てみる

```
// Palindrome.dtl（楽譜作成バッチ版）

システム！ "compose" 使う.

タイトル="Palindrome".

初期処理＝[|v_no|
  変奏番号=v_no.
  乱数初期化 (変奏番号*194612).
  速さ=(!("48,64,80,96,112,128") RandomSelect) .
  音符種類=!("2/4/8/8/8/8,2/2/4/4/4/8,2/4/4/4/8/8,2/4/8/16/16/16")
RandomSelect.
  使う音符=音符種類! "/" "," 全部置き換える.
].

作曲処理＝[|PS|
  小節数＝16. 拍子＝"4/4". 下限＝"↓ソ". 上限＝"ミ↑↑".
  音符配列＝配列! 作る. 長さ配列＝配列! 作る.
  ! (小節数)(拍子)(下限)(上限)(使う音符)(音符配列)(長さ配列)(PS) compose2.
  音符数=(音符配列! 要素数?).
  mml_1="". [|i|
    音符＝(音符配列！(i)見る)＋(長さ配列！(i)見る).
    mml_1＝mml_1＋音符＋" ".
  ]！ (音符数) 繰り返す.
  mml_2="". [|i|
    音符＝(音符配列！(音符数+1-i)見る)＋(長さ配列！(音符数+1-i)見る).
    mml_2＝mml_2＋音符＋" ".
  ]！ (音符数) 繰り返す.
].
```

```
楽譜作成＝[
  part_1＝"r2 "＋mml_1＋mml_2. part_2＝mml_1＋mml_2＋"r2".

  lly_f＝テキストファイル！ "frame.ly" 作る.
  f_wrk＝(lly_f！ 読む)! "｜" 結合.
  f_tmp＝f_wrk! "%title" (タイトル)       置き換える. f_wrk=f_tmp.
  f_tmp＝f_wrk! "%subtitle" ("No."＋変奏番号) 置き換える. f_wrk=f_tmp.
  f_tmp＝f_wrk! "%subsubtitle" (" ")      置き換える. f_wrk=f_tmp.
  f_tmp＝f_wrk! "%time" (拍子)            置き換える. f_wrk=f_tmp.
  f_tmp＝f_wrk! "%tempo" (速さ)           置き換える. f_wrk=f_tmp.
  f_tmp＝f_wrk! "%part_1" (part_1)        置き換える. f_wrk=f_tmp.
  f_tmp＝f_wrk! "%part_2" (part_2)        置き換える. f_wrk=f_tmp.
  lly_t＝テキストファイル！ (変奏番号+".ly") 作る.
  lly_t! (f_tmp! "｜" 分割) 全部書く.

  ラベル！("No."+変奏番号+" TEMPO："+速さ+" NOTE："+音符種類) 作る.
].

prf=textfile! “prm.txt” create.
変奏番号=((prf! get)! 1 get).

！(変奏番号)初期処理. ！("S") 作曲処理. ！楽譜作成.

システム! (はい) 終了する.
```

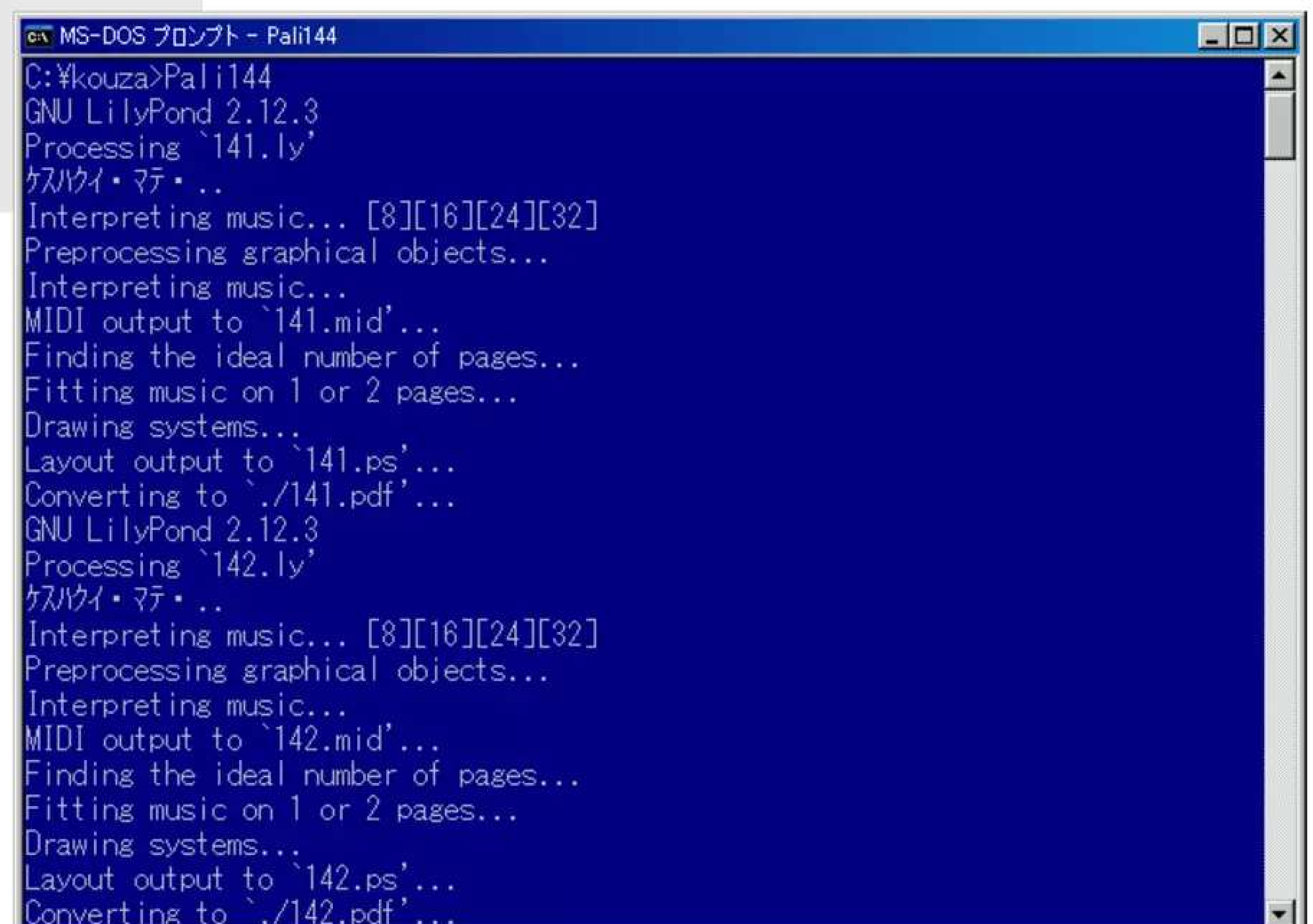
ドリトルをバッチ実行

- １曲生成するバッチ（score.bat）作る

```
@echo off
echo %1> prm.txt
dolittle.jar palindromeB.dtl
lilypond %1.ly
del %1.ps
```

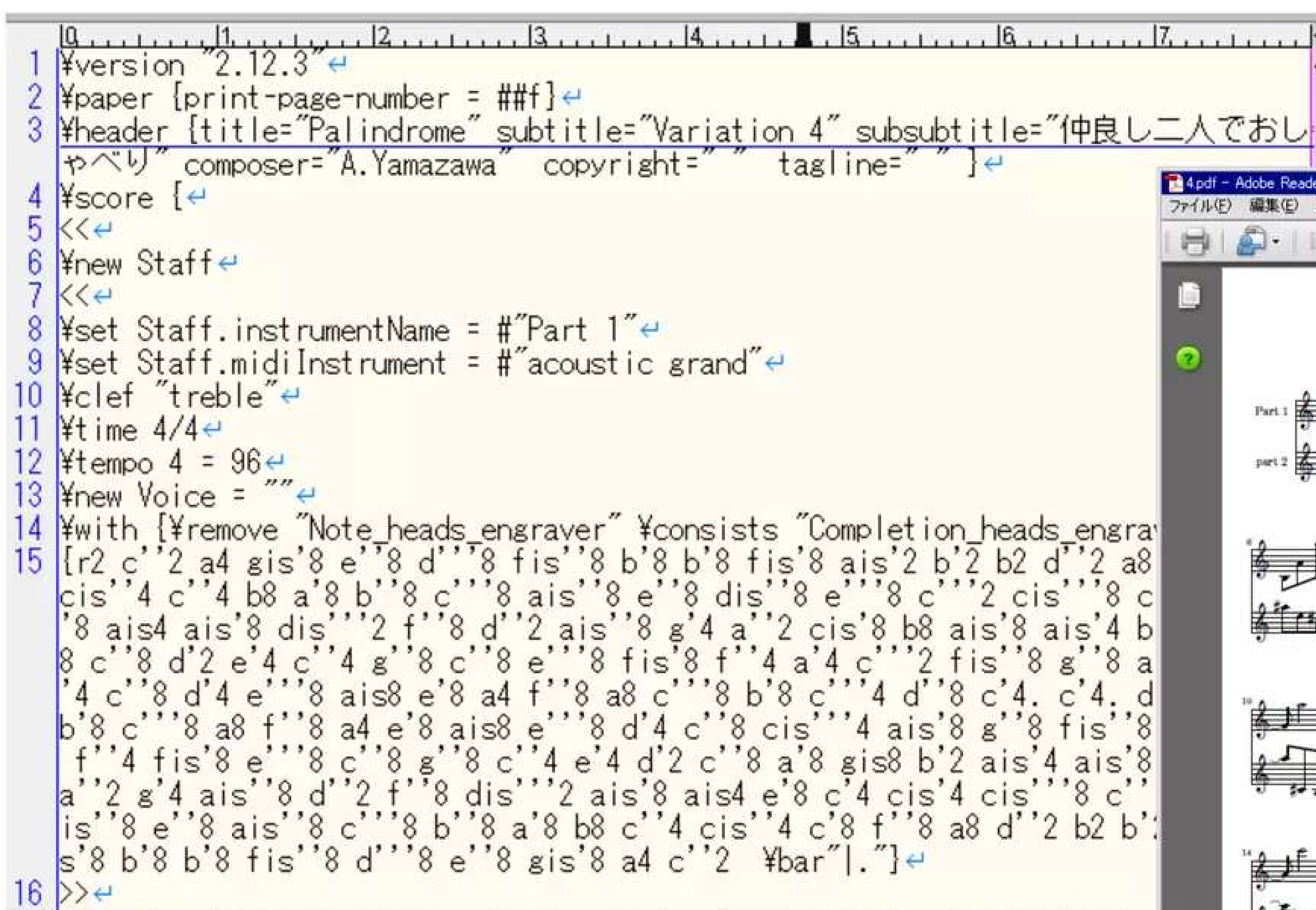
- score.batを連続実行するバッチ（Pali144.bat）を作る

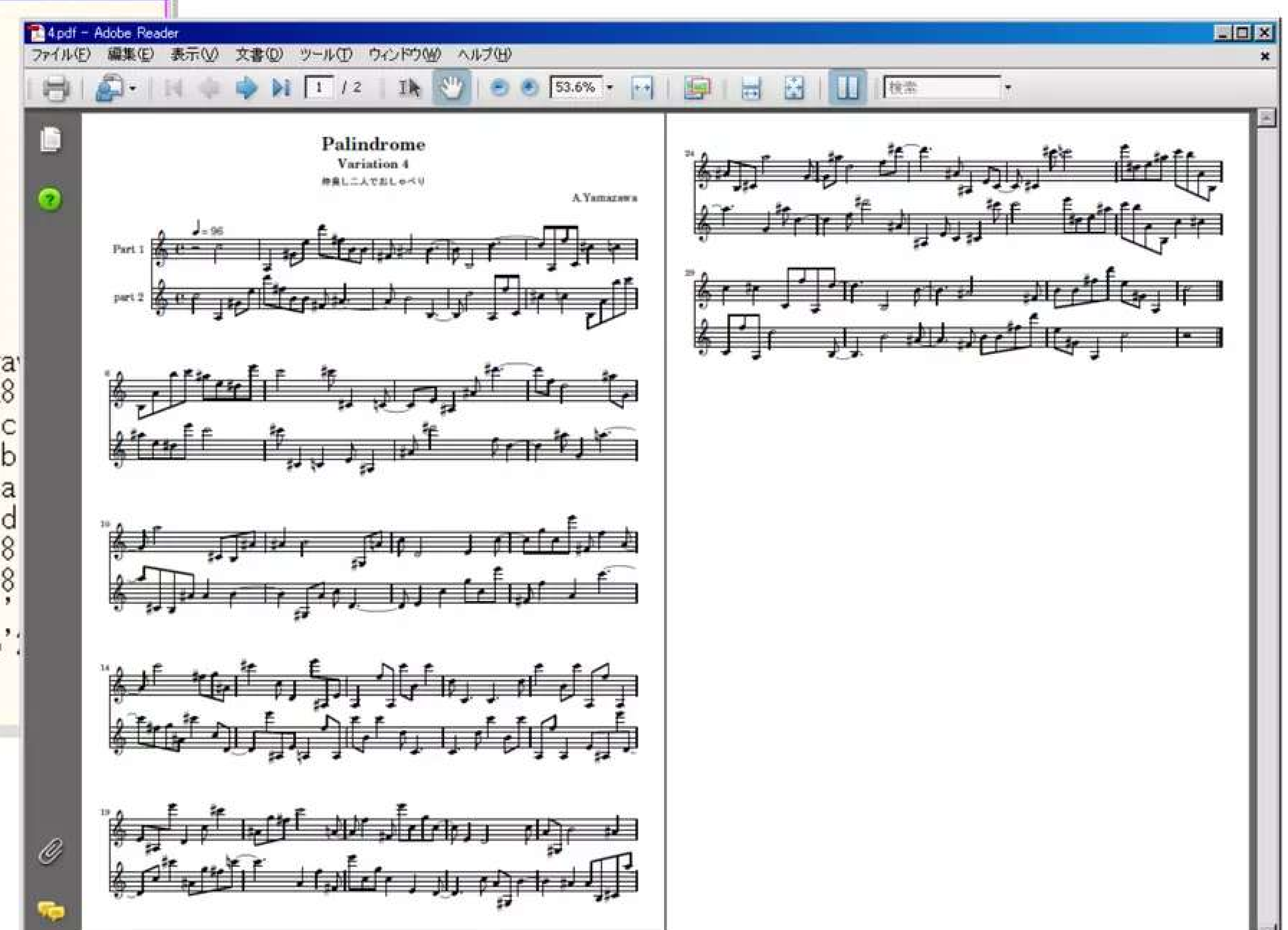
```
@echo off
call score 141
call score 142
call score 143
call score 144
```



楽譜のソースコード編集

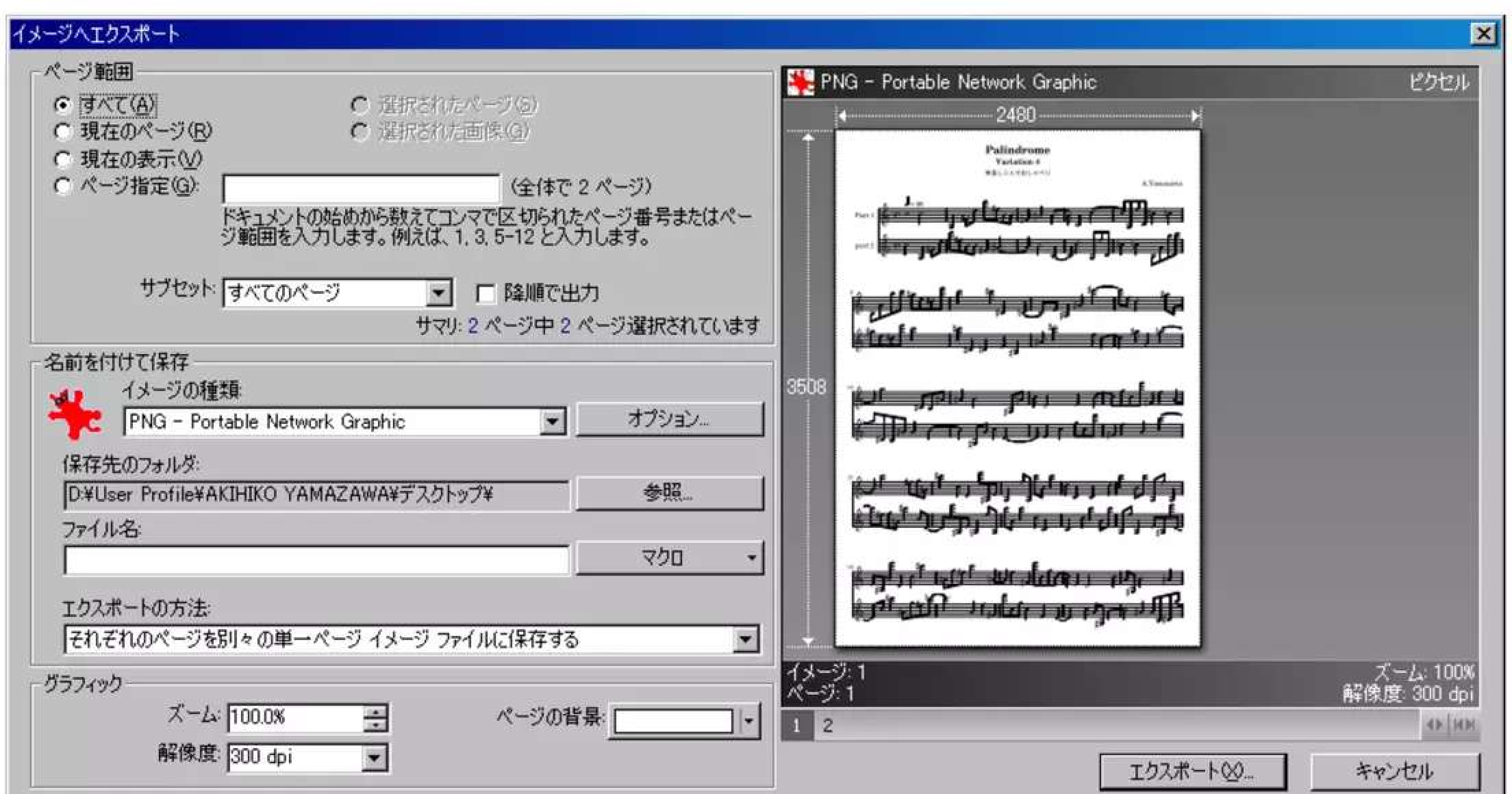
- 楽譜のソースファイル(.ly)はテキストファイル
- テキストエディタで編集できる
- タイトルの編集や強弱記号の追加などが簡単にできる
- 楽譜の調整ができたらLilyPondを実行してPDFを作成





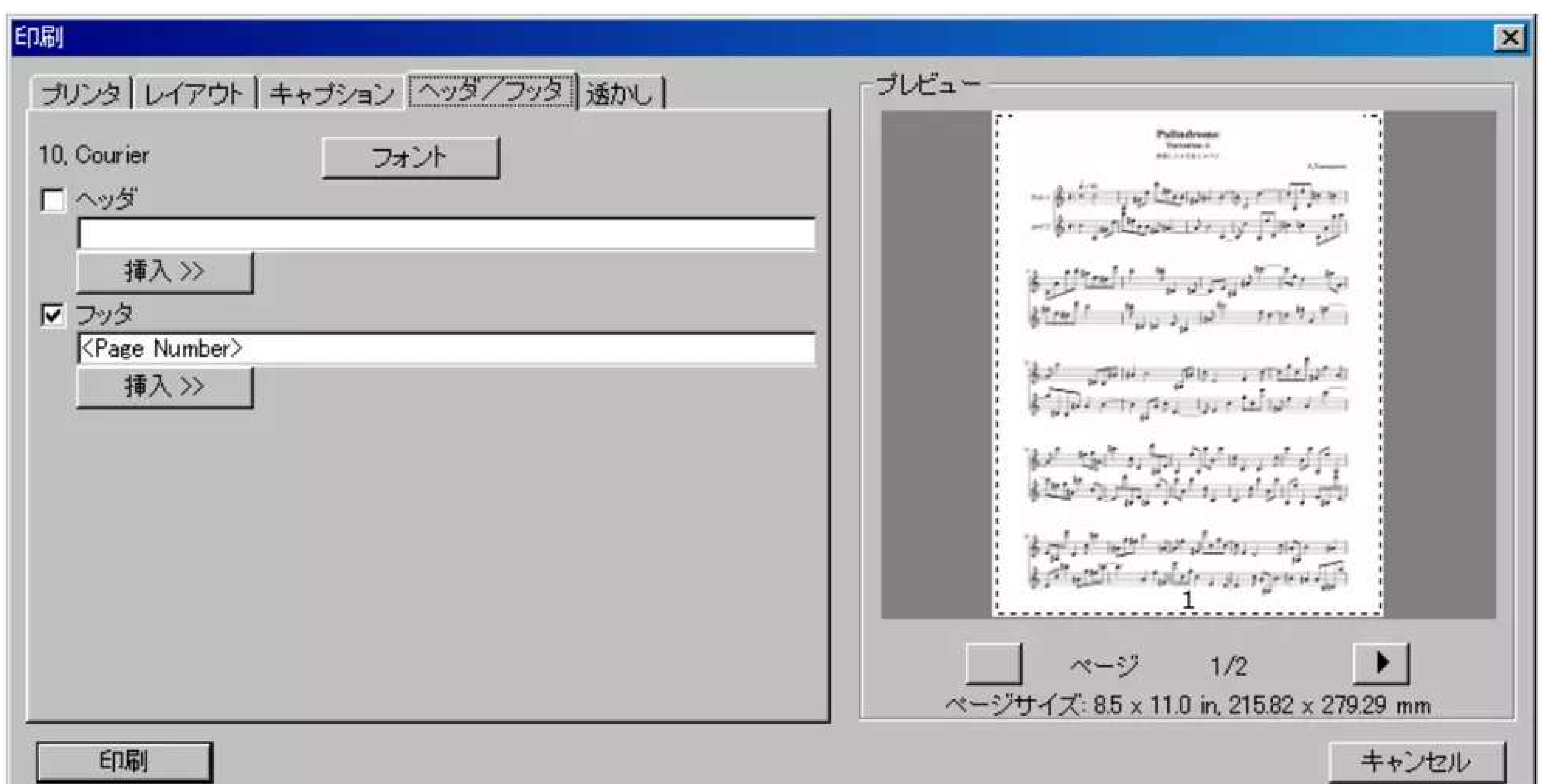
PDFを画像にする
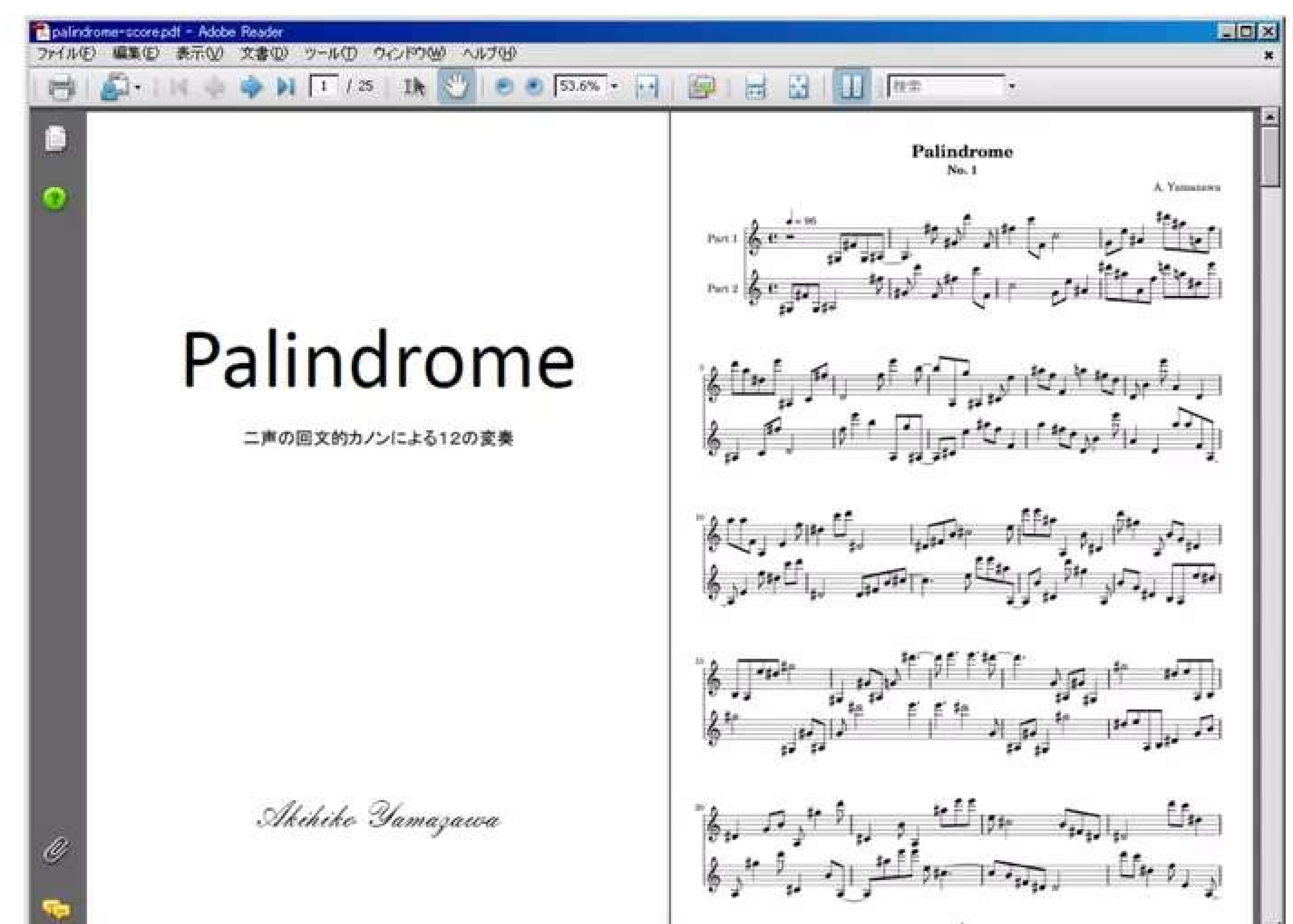
- PDFはページ毎の画像にしてしまうと出版に便利
- LilyPondeでも--pngオプションで画像を作ることが可能
- 他の方法としてPDFを画像に変換できるソフトを使う
  - PDF X-Change Viewer
  - http://www.tracker-software.com/product/downloads



楽譜を製本する

- ページ毎の画像にページ番号を付けて印刷
- 画像にページ番号を入れることができるソフト
  - XnView
  - http://www.xnview.com/
- 表紙や目次などをWriterなどを使って作成
- プリンしたものを製本すれば楽譜（曲集）の完成です

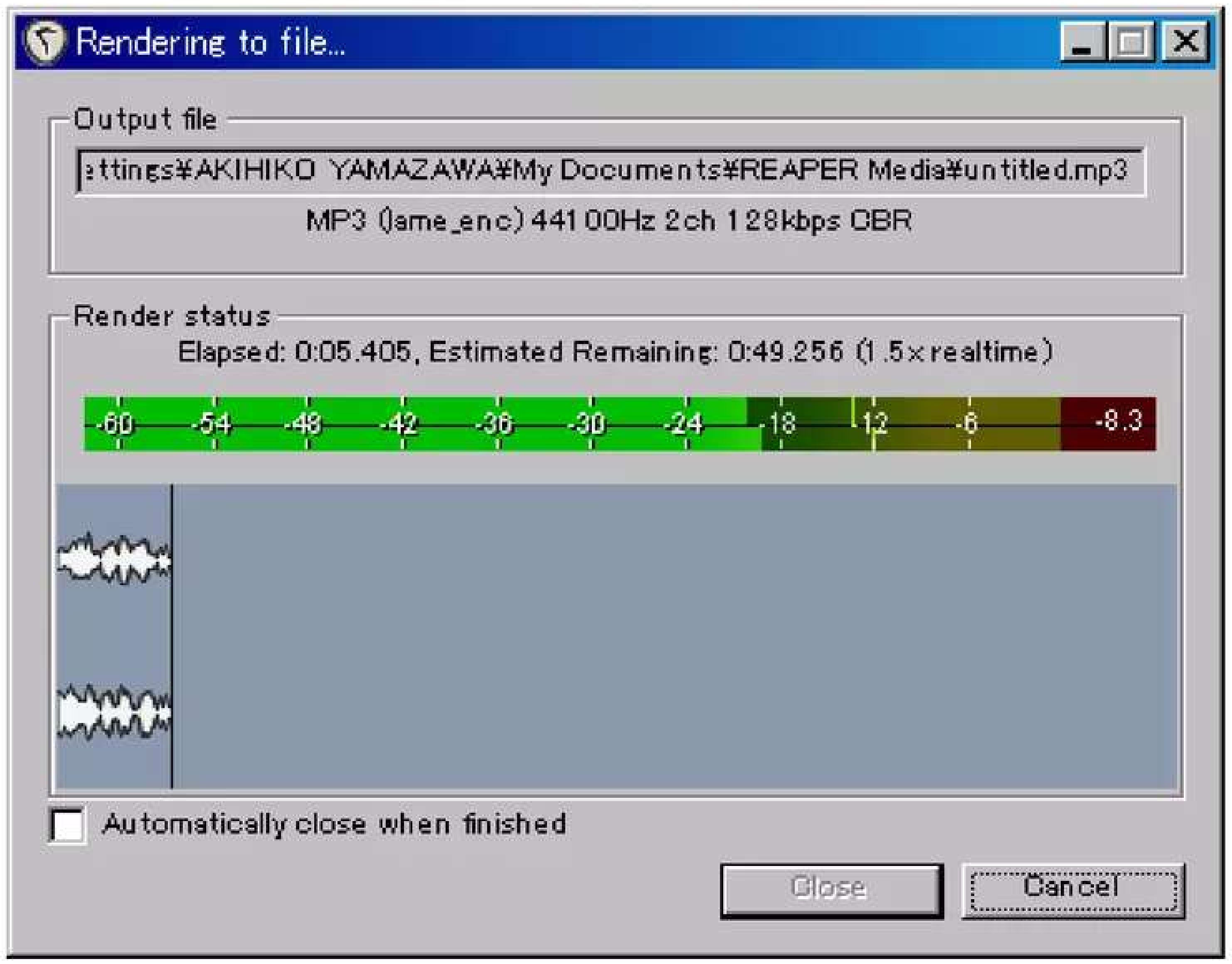




レコーディング

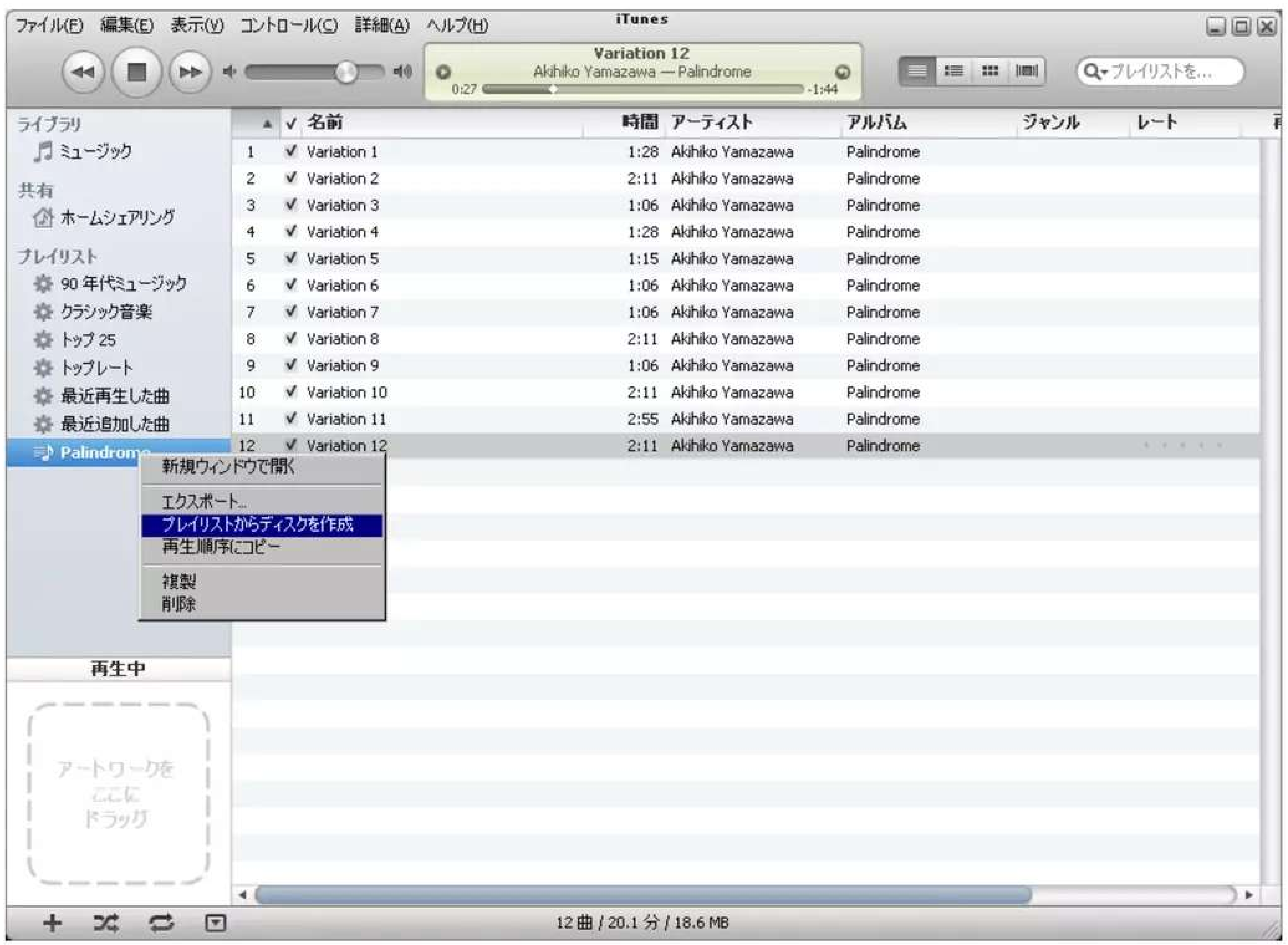
- DAWソフトを使ってMIDIデータを録音してみましょう
- 録音に使えるDAWソフト
  - REAPER
  - http://www.reaper.fm/
- MIDIデータを読み込んでトラック毎にVSTiをセットします
- 高品質のWAVやMP3ファイルを作ることができます





CDを作る

- 作成したオーディオデータでCDを制作してみましょう
- CDは iTune や Windows Media Player などで作成できます



カバーアートを作る

- CDのカバーアートも作ってみましょう
- 画像を用意してラベル印刷用のツールで編集します
- これを市販のインデックスカードに印刷します
- CDケースに収納すれば完成です

